



Authors

Gary McGraw, Ph.D., Sammy Miguez, and Jacob West

Acknowledgements

Thanks to the 78 executives from the world-class software security initiatives we studied from around the world to create BSIMM6.

They include:

| | |
|--|--|
| Adobe (Brad Arkin) | LinkedIn (Pavi Ramamurthy) |
| Aetna (Jim Routh) | Marks and Spencer |
| ANDA (Michael Tower) | McKesson (Sabastian High) |
| Autodesk (Floyd Fernandes) | NetApp (Lisa Napier) |
| Bank of America (Jim Apple) | NetSuite (Brian Chess) |
| BMO Financial Group (Richard Livesley) | Neustar (Jonathan Coombes) |
| Black Knight Financial Services (Rob Carver) | Nokia (Janne Uusilehto) |
| Box (Joel de la Garza) | NVIDIA (Eric Haller) |
| Capital One (Greg Huff) | PayPal (Erick Lee) |
| Cisco (Steve Sigel) | Pearson Learning Technologies (Aaron Weaver) |
| Citigroup (Samir Sherif) | Qualcomm (Alex Gantman) |
| Comerica Bank (George Smirnoff) | Rackspace (Jim Freeman) |
| Cryptography Research (Ron Perez) | Salesforce (Brendan O'Connor) |
| Depository Trust & Clearing Corporation (Mahi Dontamsetti) | Siemens (Jim Jacobson) |
| Elavon (Osiris Martinez) | Sony Mobile (Per-Olof Persson) |
| EMC (Eric Baize) | Symantec (Edward Bonver) |
| Epsilon (Chris Ray) | The Advisory Board (Eric Banks) |
| Experian (Suzan Nascimento) | The Home Depot (Jamil Farshchi) |
| F-Secure (Antti Vähä-Sipilä) | TomTom |
| Fannie Mae (Stephanie Derdouri) | Trainline (Mieke Kooij) |
| Fidelity | U.S. Bank (William Walker) |
| HP Fortify (Mike Donohoe) | Vanguard (Tony Canike) |
| HSBC (Malcolm Kelly and Simon Hales) | Visa (Fares Alraie) |
| Intel Security (James Ransome) | VMware (Iain Mulholland) |
| JPMorgan Chase & Co. (Jeff Cohen) | Wells Fargo (Mitch Moon) |
| Lenovo (Tony Corkell) | Zephyr Health (Kim Green) |

To those who can't be named, you know who you are, and we could not have done this without you.

Thanks to the nearly 50 individuals who helped gather the data for the BSIMM6 update. In particular, we would like to thank: Neil Bahadur, Nabil Hannan, Jason Hills, Girish Janardhanudu, Nick Murison, Kevin Nassery, Mike Ware, and Caroline Wong.

Data for The Building Security In Maturity Model was captured by Cigital. Resources for data analysis provided by NetSuite. BSIMM-V model translations produced by UTN-FRSF and Fundación Sadosky (Spanish), and Diogo Rispoli and Carolina Girardi Alves (Portuguese). BSIMM1-BSIMM3 were authored by Gary McGraw, Ph.D, Brian Chess, Ph.D, and Sammy Migues. BSIMM4 and BSIMM-V were authored by Gary McGraw, Ph.D., Sammy Migues, and Jacob West.

Executive Summary

The Building Security In Maturity Model (BSIMM) is the result of a multi-year study of real-world software security initiatives. We present the model as built directly out of data observed in 78 software security initiatives from firms including: Adobe, Aetna, ANDA, Autodesk, Bank of America, Black Knight Financial Services, BMO Financial Group, Box, Capital One, Cisco, Citigroup, Comerica, Cryptography Research, Depository Trust and Clearing Corporation, Elavon, EMC, Epsilon, Experian, Fannie Mae, Fidelity, F-Secure, HP Fortify, HSBC, Intel Security, JPMorgan Chase & Co., Lenovo, LinkedIn, Marks & Spencer, McKesson, NetApp, NetSuite, Neustar, Nokia, NVIDIA, PayPal, Pearson Learning Technologies, Qualcomm, Rackspace, Salesforce, Siemens, Sony Mobile, Symantec, The Advisory Board, The Home Depot, TomTom, Trainline, U.S. Bank, Vanguard, Visa, VMware, Wells Fargo, and Zephyr Health.

The BSIMM is a measuring stick for software security. The best way to use the BSIMM is to compare and contrast your own initiative with the data about what other organizations are doing contained in the model. You can then identify goals and objectives of your own and refer to the BSIMM to determine which additional activities make sense for you.

The BSIMM data show that high maturity initiatives are well-rounded—carrying out numerous activities in all 12 of the practices described by the model. The model also describes how mature software security initiatives evolve, change, and improve over time.

BSIMM6 License

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/legalcode> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

BSIMM6 Table of Contents

1. Part One

- a. Introduction 5
 - History
 - BSIMM6
 - Audience
 - Method
 - Participating Firms
- b. BSIMM6 Structure 10
 - The Software Security Framework
 - The BSIMM6 Skeleton
- c. Putting BSIMM6 to Use 19
 - What BSIMM Tells Us
 - Measuring Your Firm with BSIMM6
- d. BSIMM6 Analysis 25
 - BSIMM Over Time
 - The BSIMM and Industry Verticals
 - BSIMM as a Longitudinal Study
- e. BSIMM Community 31

2. Part Two

- a. Roles in a Software Security Initiative 32
 - Executive Leadership
 - Software Security Group (SSG)
 - Satellite
 - Everybody Else
- b. BSIMM6 Activities 35
 - Governance
 - 1. Strategy & Metrics (SM)
 - 2. Compliance & Policy (CP)
 - 3. Training (T)
 - Intelligence
 - 1. Attack Models (AM)
 - 2. Security Features & Design (SFD)
 - 3. Standards & Requirements (SR)
 - SSDL Touchpoints
 - 1. Architecture Analysis (AA)
 - 2. Code Review (CR)
 - 3. Security Testing (ST)
 - Deployment
 - 1. Penetration Testing (PT)
 - 2. Software Environment (SE)
 - 3. Configuration Management & Vulnerability Management (CMVM)

3. Appendix

- a. Adjusting BSIMM-V for BSIMM6 59
- b. 112 BSIMM Activities at a Glance 60

BSIMM6 List of Tables

- 1. BSIMM Terminology 9
- 2. Software Security Framework 10
- 3. BSIMM Skeleton 12
- 4. BSIMM Scorecard 21
- 5. 12 Core Activities 21
- 6. BSIMM6 Scorecard for Fake Firm 23
- 7. BSIMM Numbers Over Time 25
- 8. Vertical Comparison Scorecard 28
- 9. Longitudinal Scorecard as Measured Over Time . . 30

BSIMM6 List of Figures

- 1. Earth Spider Chart 21
- 2. BSIMM Score Distribution 22
- 3. Earth vs. Firm Spider Chart 24
- 4. Vertical Comparison Spider Chart 26
- 5. ISV vs. CE Spider Chart 27
- 6. Earth vs. Healthcare Spider Chart 27
- 7. Round 1 Earth vs. Round 2 Earth Spider Chart . . 31

PART ONE

The Building Security In Maturity Model (BSIMM, pronounced “bee simm”) is a study of existing software security initiatives. By quantifying the practices of many different organizations, we can describe the common ground shared by many as well as the variation that makes each unique. Our aim is to help the wider software security community plan, carry out, and measure initiatives of their own. The BSIMM is not a “how to” guide, nor is it a one-size-fits-all prescription. Instead, the BSIMM is a reflection of the current state of Software Security.

We begin with a brief description of the function and importance of a software security initiative. We then explain our model and the method we use for quantifying the state of an initiative. Since the BSIMM study began in 2008, we have studied 104 initiatives, which comprise 235 distinct measurements—some firms use the BSIMM to measure each of their business units and some have been measured more than once. To ensure the continued relevance of the data we report, we excluded from BSIMM6 measurements older than 42 months. The current data set comprises 202 distinct measurements collected from 78 firms. Thanks to repeat measurements, not only do we report on current practices, but also on the ways in which some initiatives have evolved over a period of years.

We devote the later portion of the document to a detailed explanation of the key roles in a software security initiative, the 112 activities that now comprise our model, and a summary of the raw data we have collected. We have reviewed the description of each activity for BSIMM6.

Our work with the BSIMM model shows that measuring a firm’s software security initiative is both possible and extremely useful. BSIMM measurements can be used to plan, structure, and execute the evolution of a software security initiative. Over time, firms participating in the BSIMM show measurable improvement in their software security initiatives.

The BSIMM is a reflection of the current state of software security.

Introduction

History

In the late 1990s, software security began to flourish as a discipline separate from computer and network security. Researchers began to put more emphasis on studying the ways a programmer can contribute to or unintentionally undermine the security of a computer system: What kinds of bugs and flaws lead to security problems? How can we identify problems systematically?

By the middle of the following decade, there was an emerging consensus that creating secure software required more than just smart individuals toiling away. Getting security right means being involved in the software development process.

Since then, practitioners have come to know that process alone is insufficient. Software security encompasses business, social, and organizational aspects as well. We use the term software security initiative (SSI) to refer to all of the activities undertaken for the purpose of building secure software.

BSIMM6

The purpose of the BSIMM is to quantify the activities carried out by real software security initiatives. Because these initiatives use different methodologies and different terminology, the BSIMM requires a framework that allows us to describe all of the initiatives in a uniform way. Our software security framework (SSF) and activity descriptions provide a common vocabulary for explaining the salient elements of a software security initiative, thereby allowing us to compare initiatives that use different terms, operate at different scales, exist in different vertical markets, or create different work products.

We classify our work as a maturity model because improving software security almost always means changing the way an organization works—something that doesn't happen overnight. We understand that not all organizations need to achieve the same security goals, but we believe all organizations can benefit from using the same measuring stick.

**We believe all
organizations can benefit
from using the same
measuring stick.**

BSIMM6 is the sixth major version of the BSIMM model. It includes updated activity descriptions, data from 78 firms in multiple vertical markets, and a longitudinal study.

Audience

The BSIMM is meant for use by anyone responsible for creating and executing an SSI. We have observed that successful software security initiatives are typically run by a senior executive who reports to the highest levels in an organization. These executives lead an internal group that we call the software security group (SSG), charged with directly executing or facilitating the activities described in the BSIMM. The BSIMM is written with the SSG and SSG leadership in mind.

We expect readers to be familiar with the software security literature. You can become familiar with many concepts by reading [Software Security: Building Security In](#). The BSIMM does not attempt to explain software security basics, describe its history, or provide references to the ever-expanding literature. Succeeding with the BSIMM without becoming familiar with the literature is unlikely.

Method

We built the first version of the BSIMM in Fall of 2008 as follows:

- We relied on our own knowledge of software security practices to create the SSF (we present the framework on page 10).
- We conducted a series of in-person interviews with nine executives in charge of software security initiatives. From these interviews, we identified a set of common activities, which we organized according to the SSF.
- We then created scorecards for each of the nine initiatives that show which activities the initiatives carry out. To validate our work, we asked each participating firm to review the framework, the practices, and the scorecard we created for their initiative.

The BSIMM is a data-driven model that evolves over time. We have added, deleted, and adjusted the level of various activities based on the data observed as the project has evolved. To preserve backwards compatibility, all changes are made by adding new activity labels to the model, even when an activity has simply changed levels. We make changes by considering outliers both in the model itself and in the levels we assigned to various activities in the 12 practices. We use the results of an intra-level standard deviation analysis to determine which “outlier” activities to move between levels. We focus on changes that minimize standard deviation in the average number of observed activities at each level.

In some rare instances, we have observed an activity in the field that is not yet captured in the model. Our criteria for adding an activity to the BSIMM are as follows: If we observe a candidate activity not yet in the model, we determine (based on previously captured data and queries to the BSIMM mailing list) how many firms appear to carry out that activity. If the answer is multiple firms, we take a closer look at the proposed activity and figure out how it fits with the existing model. If the answer is only one firm, the candidate activity is tabled as too specialized.

We used an in-person interview technique to conduct BSIMM assessments for a total of 104 firms so far. In 13 cases, we assessed the SSG and multiple business units as part of creating the corporate view of their SSI. In some cases, we used one aggregated scorecard while in other cases we used multiple scorecards for the SSG and each business unit. Each firm is represented by only one set of data in the model published here. Beginning with BSIMM-V, we introduced a data freshness requirement to exclude measurements older than 48 months. For BSIMM6, we lowered this data freshness threshold to 42 months. This requirement caused 26 firms to be removed from the BSIMM data, resulting in the BSIMM6 data set representing 78 firms. As our study progresses, we intend to decrease the freshness window to 36 months to better align with business cycles.

We used the resulting scores to refine the set of activities and their placement in the framework. We have also conducted a second complete set of interviews with 26 of the current participating firms in order to study how their initiatives have changed over time. Ten firms have undertaken three BSIMM assessments, two have done four BSIMM assessments, and one firm has had five BSIMM assessments.

We hold the scorecards for individual firms in confidence, but we publish aggregate data describing the number of times we have observed each activity (see page 20). We also publish observations about subsets (such as industry verticals) when our sample size for the subset is large enough to guarantee the anonymity of the firms.

As a descriptive model, the only goal of the BSIMM is to observe and report. We like to say that we wandered off into the jungle to see what we could see and discovered that “monkeys eat bananas in X of the Y jungles we visited.” Notice that the BSIMM does not report, “you should only eat yellow bananas,” “do not run while eating a banana,”

“thou shalt not steal thy neighbors’ bananas,” or any other value judgments. Simple observations, simply reported.

Our “just the facts” approach is hardly novel in science and engineering, but in the realm of software security it has not previously been applied at this scale. Previous work has either described the experience of a single organization or offered prescriptive guidance based only on a combination of personal experience and opinion.

**Simple observations,
simply reported.**

Participating Firms

The 78 participating organizations are drawn from four well-represented verticals (with some overlap): financial services (33), independent software vendors (27), consumer electronics (13), and healthcare (10). Verticals with lower representation in the BSIMM population include: insurance, telecommunications, security, retail, and energy.

Those companies among the 78 who graciously agreed to be identified include:

Adobe, Aetna, ANDA, Autodesk, Bank of America, Black Knight Financial Services, BMO Financial Group, Box, Capital One, Cisco, Citigroup, Comerica, Cryptography Research, Depository Trust and Clearing Corporation, Elavon, EMC, Epsilon, Experian, Fannie Mae, Fidelity, F-Secure, HP Fortify, HSBC, Intel Security, JPMorgan Chase & Co., Lenovo, LinkedIn, Marks & Spencer, McKesson, NetApp, NetSuite, Neustar, Nokia, NVIDIA, PayPal, Pearson Learning Technologies, Qualcomm, Rackspace, Salesforce, Siemens, Sony Mobile, Symantec, The Advisory Board, The Home Depot, TomTom, trainline, U.S. Bank, Vanguard, Visa, VMware, Wells Fargo, and Zephyr Health

On average, the 78 participating firms had practiced software security for 3.98 years at the time of assessment (ranging from less than a year old to 15 years old as of October, 2015). All 78 firms agree that the success of their initiative hinges on having an internal group devoted to software security—the SSG. SSG size on average is 13.9 people (smallest 1, largest 130, median 6) with a “satellite” of others (developers, architects, and people in the organization directly engaged in and promoting software security) of 27.1 people (smallest 0, largest 400, median 3). The average number of developers among our targets was 3,680 people (smallest 23, largest 35,000, median 1,200), yielding an average percentage of SSG to development of 1.51% (median 0.7%).

All told, the BSIMM describes the work of 1,084 SSG members working with a satellite of 2,111 people to secure the software developed by 287,006 developers.

BSIMM Terminology

Nomenclature has always been a problem in computer security and software security is no exception. Several terms used in the BSIMM have particular meaning for us. Here are some of the most important terms used throughout the document:

Activity: Actions carried out or facilitated by the SSG as part of a practice. Activities are divided into three levels in the BSIMM.

Domain: The domains are: governance, intelligence, secure software development lifecycle (SSDL) touchpoints, and deployment. See the SSF section on page 10.

Practice: One of the 12 categories of BSIMM activities. Each domain in the Software Security Framework has three practices. Activities in each practice are divided into three levels. See the SSF section on page 10.

Satellite: A group of interested and engaged developers, architects, software managers, testers, and similar roles who have a natural affinity for software security and are organized and leveraged by a Software Security Initiative.

Secure Software Development Lifecycle (SSDL): Any SDLC with integrated software security checkpoints and activities.

Security Development Lifecycle (SDL): A term used by Microsoft to describe their Secure Software Development Lifecycle.

Software Security Framework (SSF): The basic structure underlying the BSIMM, comprising 12 practices divided into four domains. See the SSF section on page 10.

Software Security Group (SSG): The internal group charged with carrying out and facilitating software security. According to our observations, the first step of a Software Security Initiative is forming an SSG.

Software Security Initiative: An organization-wide program to instill, measure, manage, and evolve software security activities in a coordinated fashion. Also known in the literature as an Enterprise Software Security Program (see chapter 10 of [Software Security: Building Security In](#)).





BSIMM6 Structure

The BSIMM is organized as a set of 112 activities in a framework.

The Software Security Framework

The table below shows the software security framework (SSF) used to organize the 112 BSIMM activities. There are 12 practices organized into four domains.

The four domains are:

- **Governance:** Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.
- **Intelligence:** Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modeling.
- **SSDL Touchpoints:** Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.
- **Deployment:** Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance and other environment issues have direct impact on software security.

The 12 practices are:

| | |
|------------------|--|
| Governance | <div>1. Strategy & Metrics (SM) 2. Compliance & Policy (CP) 3. Training (T)</div> |
| Intelligence | <div>4. Attack Models (AM) 5. Security Features & Design (SFD) 6. Standards & Requirements (SR)</div> |
| SSDL Touchpoints | <div>7. Architecture Analysis (AA) 8. Code Review (CR) 9. Security Testing (ST)</div> |
| Deployment | <div>10. Penetration Testing (PT) 11. Software Environment (SE) 12. Configuration Management & Vulnerability Management (CMVM)</div> |



BSIMM 6

The BSIMM6 Skeleton

The BSIMM skeleton provides a way to view the model at a glance and is useful when assessing a software security initiative. The skeleton is shown below, organized by practices and levels, and the percentage of the firms, out of 78, performing that activity in their SSI. More complete descriptions of the activities, examples, and term definitions can be found in Part Two of this document.



Governance

STRATEGY & METRICS (SM)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Publish process (roles, responsibilities, plan), evolve as necessary. | SM1.1 | 53% |
| Create evangelism role and perform internal marketing. | SM1.2 | 51% |
| Educate executives. | SM1.3 | 46% |
| Identify gate locations, gather necessary artifacts. | SM1.4 | 85% |

LEVEL 2

| | | |
|---|-------|-----|
| Publish data about software security internally. | SM2.1 | 46% |
| Enforce gates with measurements and track exceptions. | SM2.2 | 37% |
| Create or grow a satellite. | SM2.3 | 38% |
| Identify metrics and use them to drive budgets. | SM2.5 | 22% |
| Require security sign-off. | SM2.6 | 37% |

LEVEL 3

| | | |
|---|-------|-----|
| Use an internal tracking application with portfolio view. | SM3.1 | 19% |
| Run an external marketing program. | SM3.2 | 9% |

COMPLIANCE & POLICY (CP)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|-----------------------------|------------|---------------|
| Unify regulatory pressures. | CP1.1 | 58% |
| Identify PII obligations. | CP1.2 | 78% |
| Create policy. | CP1.3 | 53% |

Table continued on next page >



Governance continued...

LEVEL 2

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Identify PII data inventory. | CP2.1 | 24% |
| Require security sign-off for compliance-related risk. | CP2.2 | 29% |
| Implement and track controls for compliance. | CP2.3 | 32% |
| Paper all vendor contracts with software security SLAs. | CP2.4 | 37% |
| Ensure executive awareness of compliance and privacy obligations. | CP2.5 | 42% |

LEVEL 3

| | | |
|---|-------|-----|
| Create regulator eye-candy. | CP3.1 | 23% |
| Impose policy on vendors. | CP3.2 | 14% |
| Drive feedback from SSDL data back to policy. | CP3.3 | 8% |

TRAINING (T)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Provide awareness training. | T1.1 | 76% |
| Deliver role-specific advanced curriculum (tools, technology stacks, bug parade). | T1.5 | 33% |
| Create and use material specific to company history. | T1.6 | 22% |
| Deliver on-demand individual training. | T1.7 | 46% |

LEVEL 2

| | | |
|--|------|-----|
| Enhance satellite through training and events. | T2.5 | 13% |
| Include security resources in onboarding. | T2.6 | 19% |
| Identify satellite through training. | T2.7 | 8% |

LEVEL 3

| | | |
|--|------|-----|
| Reward progression through curriculum (certification or HR). | T3.1 | 4% |
| Provide training for vendors or outsourced workers. | T3.2 | 4% |
| Host external software security events. | T3.3 | 4% |
| Require an annual refresher. | T3.4 | 10% |
| Establish SSG office hours. | T3.5 | 5% |

ATTACK MODELS (AM)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|--|------------|---------------|
| Build and maintain a top N possible attacks list. | AM1.1 | 22% |
| Create a data classification scheme and inventory. | AM1.2 | 65% |
| Identify potential attackers. | AM1.3 | 40% |
| Collect and publish attack stories. | AM1.4 | 10% |
| Gather and use attack intelligence. | AM1.5 | 59% |
| Build an internal forum to discuss attacks. | AM1.6 | 14% |

LEVEL 2

| | | |
|--|-------|-----|
| Build attack patterns and abuse cases tied to potential attackers. | AM2.1 | 8% |
| Create technology-specific attack patterns. | AM2.2 | 10% |

LEVEL 3

| | | |
|---|-------|----|
| Have a science team that develops new attack methods. | AM3.1 | 5% |
| Create and use automation to do what attackers will do. | AM3.2 | 3% |

SECURITY FEATURES & DESIGN (SFD)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|--------------------------------------|------------|---------------|
| Build and publish security features. | SFD1.1 | 78% |
| Engage SSG with architecture. | SFD1.2 | 76% |

LEVEL 2

| | | |
|--|--------|-----|
| Build secure-by-design middleware frameworks and common libraries. | SFD2.1 | 31% |
| Create SSG capability to solve difficult design problems. | SFD2.2 | 50% |

LEVEL 3

| | | |
|--|--------|-----|
| Form a review board or central committee to approve and maintain secure design patterns. | SFD3.1 | 10% |
| Require use of approved security features and frameworks. | SFD3.2 | 14% |
| Find and publish mature design patterns from the organization. | SFD3.3 | 3% |



Intelligence continued...

STANDARDS & REQUIREMENTS (SR)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Create security standards. | SR1.1 | 73% |
| Create a security portal. | SR1.2 | 64% |
| Translate compliance constraints to requirements. | SR1.3 | 67% |

LEVEL 2

| | | |
|---|-------|-----|
| Create a standards review board. | SR2.2 | 35% |
| Create standards for technology stacks. | SR2.3 | 27% |
| Identify open source. | SR2.4 | 24% |
| Create SLA boilerplate. | SR2.5 | 26% |
| Use secure coding standards. | SR2.6 | 29% |

LEVEL 3

| | | |
|-----------------------------------|-------|-----|
| Control open source risk. | SR3.1 | 8% |
| Communicate standards to vendors. | SR3.2 | 14% |

ARCHITECTURE ANALYSIS (AA)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Perform security feature review. | AA1.1 | 86% |
| Perform design review for high-risk applications. | AA1.2 | 37% |
| Have SSG lead design review efforts. | AA1.3 | 28% |
| Use a risk questionnaire to rank applications. | AA1.4 | 59% |

LEVEL 2

| | | |
|---|-------|-----|
| Define and use AA process. | AA2.1 | 15% |
| Standardize architectural descriptions (including data flow). | AA2.2 | 12% |
| Make SSG available as AA resource or mentor. | AA2.3 | 17% |

LEVEL 3

| | | |
|---|-------|----|
| Have software architects lead design review efforts. | AA3.1 | 8% |
| Drive analysis results into standard architecture patterns. | AA3.2 | 1% |

CODE REVIEW (CR)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Use a top N bugs list (real data preferred). | CR1.1 | 23% |
| Have SSG perform ad hoc review. | CR1.2 | 68% |
| Use automated tools along with manual review. | CR1.4 | 71% |
| Make code review mandatory for all projects. | CR1.5 | 31% |
| Use centralized reporting to close the knowledge loop and drive training. | CR1.6 | 35% |

LEVEL 2

| | | |
|--|-------|-----|
| Enforce coding standards. | CR2.2 | 9% |
| Assign tool mentors. | CR2.5 | 26% |
| Use automated tools with tailored rules. | CR2.6 | 21% |

LEVEL 3

| | | |
|--|-------|----|
| Build a factory. | CR3.2 | 4% |
| Build a capability for eradicating specific bugs from the entire codebase. | CR3.3 | 6% |
| Automate malicious code detection. | CR3.4 | 4% |

</> SSDL Touchpoints continued...

SECURITY TESTING (ST)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Ensure QA supports edge/boundary value condition testing. | ST1.1 | 78% |
| Drive tests with security requirements and security features. | ST1.3 | 85% |

LEVEL 2

| | | |
|---|-------|-----|
| Integrate black box security tools into the QA process. | ST2.1 | 31% |
| Share security results with QA. | ST2.4 | 10% |
| Include security tests in QA automation. | ST2.5 | 13% |
| Perform fuzz testing customized to application APIs. | ST2.6 | 14% |

LEVEL 3

| | | |
|--|-------|----|
| Drive tests with risk analysis results. | ST3.3 | 5% |
| Leverage coverage analysis. | ST3.4 | 5% |
| Begin to build and apply adversarial security tests (abuse cases). | ST3.5 | 6% |

PENETRATION TESTING (PT)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|--|------------|---------------|
| Use external penetration testers to find problems. | PT1.1 | 88% |
| Feed results to the defect management and mitigation system. | PT1.2 | 60% |
| Use penetration testing tools internally. | PT1.3 | 60% |

LEVEL 2

| | | |
|---|-------|-----|
| Provide penetration testers with all available information. | PT2.2 | 26% |
| Schedule periodic penetration tests for application coverage. | PT2.3 | 22% |

LEVEL 3

| | | |
|---|-------|-----|
| Use external penetration testers to perform deep-dive analysis. | PT3.1 | 13% |
| Have the SSG customize penetration testing tools and scripts. | PT3.2 | 10% |

SOFTWARE ENVIRONMENT (SE)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Use application input monitoring. | SE1.1 | 47% |
| Ensure host and network security basics are in place. | SE1.2 | 88% |

LEVEL 2

| | | |
|------------------------------|-------|-----|
| Publish installation guides. | SE2.2 | 40% |
| Use code signing. | SE2.4 | 32% |

LEVEL 3

| | | |
|--|-------|-----|
| Use code protection. | SE3.2 | 13% |
| Use application behavior monitoring and diagnostics. | SE3.3 | 6% |



Deployment continued...

CONFIGURATION MANAGEMENT & VULNERABILITY MANAGEMENT (CMVM)

LEVEL 1

| ACTIVITY DESCRIPTION | ACTIVITY # | PARTICIPANT % |
|---|------------|---------------|
| Create or interface with incident response. | CMVM1.1 | 91% |
| Identify software defects found in operations monitoring and feed them back to development. | CMVM1.2 | 94% |

LEVEL 2

| | | |
|--|---------|-----|
| Have emergency codebase response. | CMVM2.1 | 82% |
| Track software bugs found in operations through the fix process. | CMVM2.2 | 78% |
| Develop an operations inventory of applications. | CMVM2.3 | 40% |

LEVEL 3

| | | |
|--|---------|----|
| Fix all occurrences of software bugs found in operations. | CMVM3.1 | 5% |
| Enhance the SSDL to prevent software bugs found in operations. | CMVM3.2 | 8% |
| Simulate software crisis. | CMVM3.3 | 8% |
| Operate a bug bounty program. | CMVM3.4 | 4% |

Putting BSIMM6 to Use

The BSIMM describes 112 activities that any organization can put into practice. The activities are structured in terms of the SSF, which identifies 12 practices grouped into four domains.

What BSIMM6 Tells Us

The BSIMM data yield very interesting analytical results. The BSIMM6 scorecard on the following page shows the number of times each of the 112 activities briefly outlined in the BSIMM skeleton was observed in the BSIMM6 data. This is the highest resolution BSIMM data that is published.

“I’m so glad to see this important body of work continuing to grow and evolve. BSIMM remains one of the best yardsticks available to practitioners today for measuring how their secure software development stacks up against the rest of the industry.”

– Kenneth R. van Wyk, President, KRvW

BSIMM6 SCORECARD

| GOVERNANCE | | INTELLIGENCE | | SSDL TOUCHPOINTS | | DEPLOYMENT | |
|------------|----------|--------------|----------|------------------|----------|------------|----------|
| ACTIVITY | OBSERVED | ACTIVITY | OBSERVED | ACTIVITY | OBSERVED | ACTIVITY | OBSERVED |
| [SM1.1] | 41 | [AM1.1] | 17 | [AA1.1] | 67 | [PT1.1] | 69 |
| [SM1.2] | 40 | [AM1.2] | 51 | [AA1.2] | 29 | [PT1.2] | 47 |
| [SM1.3] | 36 | [AM1.3] | 31 | [AA1.3] | 22 | [PT1.3] | 47 |
| [SM1.4] | 66 | [AM1.4] | 8 | [AA1.4] | 46 | [PT2.2] | 20 |
| [SM2.1] | 36 | [AM1.5] | 46 | [AA2.1] | 12 | [PT2.3] | 17 |
| [SM2.2] | 29 | [AM1.6] | 11 | [AA2.2] | 9 | [PT3.1] | 10 |
| [SM2.3] | 30 | [AM2.1] | 6 | [AA2.3] | 13 | [PT3.2] | 8 |
| [SM2.5] | 17 | [AM2.2] | 8 | [AA3.1] | 6 | | |
| [SM2.6] | 29 | [AM3.1] | 4 | [AA3.2] | 1 | | |
| [SM3.1] | 15 | [AM3.2] | 2 | | | | |
| [SM3.2] | 7 | | | | | | |
| [CP1.1] | 45 | [SFD1.1] | 61 | [CR1.1] | 18 | [SE1.1] | 37 |
| [CP1.2] | 61 | [SFD1.2] | 59 | [CR1.2] | 53 | [SE1.2] | 69 |
| [CP1.3] | 41 | [SFD2.1] | 24 | [CR1.4] | 55 | [SE2.2] | 31 |
| [CP2.1] | 19 | [SFD2.2] | 39 | [CR1.5] | 24 | [SE2.4] | 25 |
| [CP2.2] | 23 | [SFD3.1] | 8 | [CR1.6] | 27 | [SE3.2] | 10 |
| [CP2.3] | 25 | [SFD3.2] | 11 | [CR2.2] | 7 | [SE3.3] | 5 |
| [CP2.4] | 29 | [SFD3.3] | 2 | [CR2.5] | 20 | | |
| [CP2.5] | 33 | | | [CR2.6] | 16 | | |
| [CP3.1] | 18 | | | [CR3.2] | 3 | | |
| [CP3.2] | 11 | | | [CR3.3] | 5 | | |
| [CP3.3] | 6 | | | [CR3.4] | 3 | | |
| [T1.1] | 59 | [SR1.1] | 57 | [ST1.1] | 61 | [CMVM1.1] | 71 |
| [T1.5] | 26 | [SR1.2] | 50 | [ST1.3] | 66 | [CMVM1.2] | 73 |
| [T1.6] | 17 | [SR1.3] | 52 | [ST2.1] | 24 | [CMVM2.1] | 64 |
| [T1.7] | 36 | [SR2.2] | 27 | [ST2.4] | 8 | [CMVM2.2] | 61 |
| [T2.5] | 10 | [SR2.3] | 21 | [ST2.5] | 10 | [CMVM2.3] | 31 |
| [T2.6] | 15 | [SR2.4] | 19 | [ST2.6] | 11 | [CMVM3.1] | 4 |
| [T2.7] | 6 | [SR2.5] | 20 | [ST3.3] | 4 | [CMVM3.2] | 6 |
| [T3.1] | 3 | [SR2.6] | 23 | [ST3.4] | 4 | [CMVM3.3] | 6 |
| [T3.2] | 3 | [SR3.1] | 6 | [ST3.5] | 5 | [CMVM3.4] | 3 |
| [T3.3] | 3 | [SR3.2] | 11 | | | | |
| [T3.4] | 8 | | | | | | |
| [T3.5] | 4 | | | | | | |

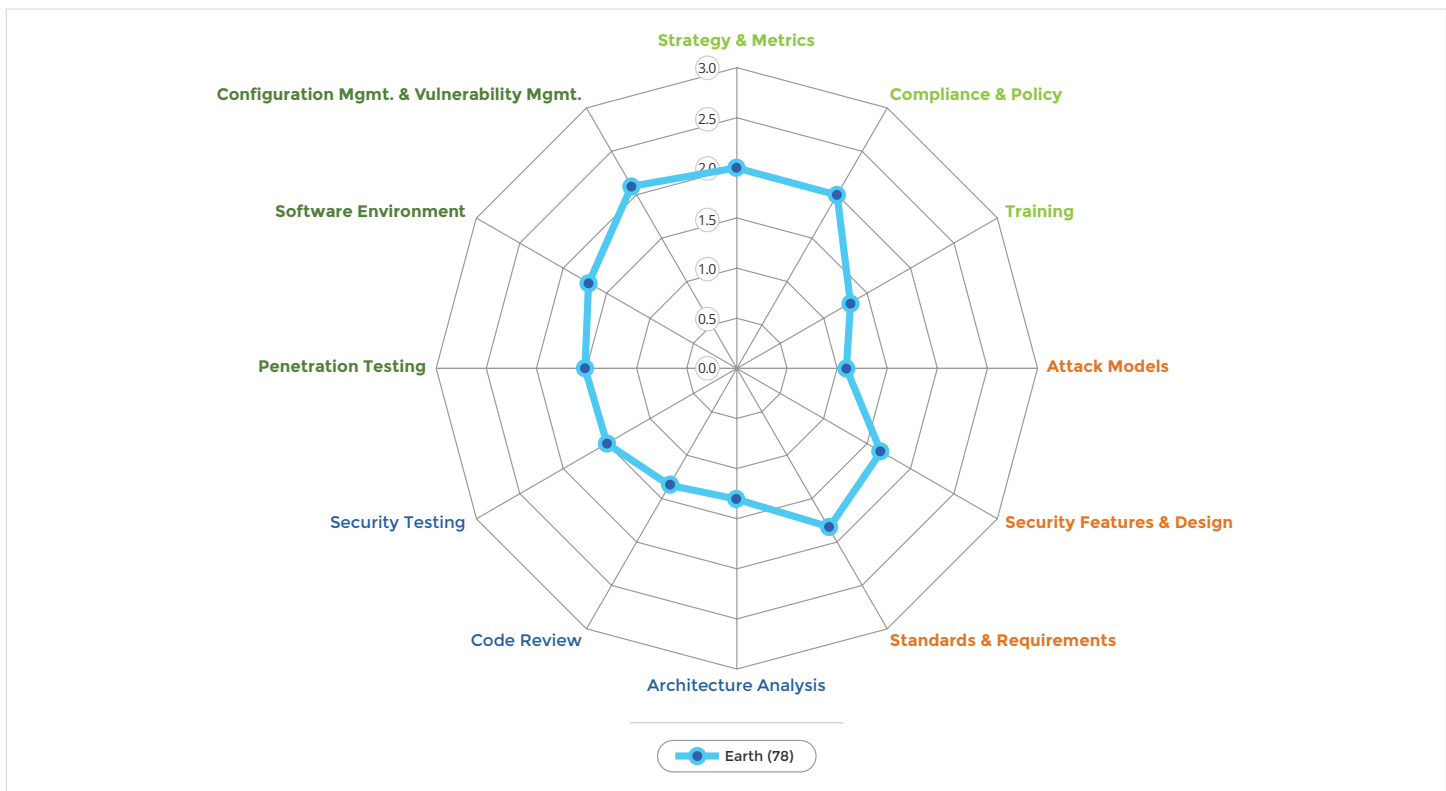
In the table above, we also identified the most common activity in each practice (shown in yellow in the scorecard). These 12 activities were observed in at least 51 (64%) of the 78 firms we studied. Though we can't directly conclude that these 12 activities are necessary for all software security initiatives, we can say with confidence that these activities are commonly found in highly successful programs. This suggests that if you are working on an initiative of your own, you should consider these 12 activities particularly carefully (not to mention the other 100).

TWELVE CORE ACTIVITIES “EVERYBODY” DOES

| ACTIVITY | DESCRIPTION |
|-----------|---|
| [SM1.4] | Identify gate locations and gather necessary artifacts |
| [CP1.2] | Identify PII obligations |
| [T1.1] | Provide awareness training |
| [AM1.2] | Create a data classification scheme and inventory |
| [SFD1.1] | Build and publish security features |
| [SR1.1] | Create security standards |
| [AA1.1] | Perform security feature review |
| [CR1.4] | Use automated tools along with manual review |
| [ST1.3] | Drive tests with security requirements and security features |
| [PT1.1] | Use external penetration testers to find problems |
| [SE1.2] | Ensure host and network security basics are in place |
| [CMVM1.2] | Identify software bugs found in operations monitoring and feed them back to development |

Spider charts are created by noting the highest-level activity for each practice per BSIMM firm (a “high-water mark”) and averaging these scores over a group of firms to produce 12 numbers (one for each practice). The resulting spider chart plots these values on 12 activity spokes corresponding to the 12 practices. Note that level 3 (the outside edge) is considered more mature than level 0 (center point). Other, more sophisticated analyses are possible, of course. We continue to experiment with weightings by level, normalization by number of activities and other schemes.

EARTH SPIDER CHART



By computing a score for each firm in the study, we can also compare relative and average maturity for one firm against the others. The range of observed scores in the current data pool is [10, 85].

The graph below shows the distribution of scores among the population of 78 participating firms. To make this graph, we divided the scores into six bins. As you can see, the scores represent a slightly skewed bell curve. We also plotted the average age of the firms in each bin as the orange line on the graph. In general, firms where more BSIMM activities have been observed have older software security initiatives.

SCORE DISTRIBUTION



We are pleased that the BSIMM study continues to grow year after year (the data set we report on here is more than 26 times the size it was for the original publication). Note that once we exceeded a sample size of 30 firms, we began to apply statistical analysis, yielding statistically significant results.

Measuring Your Firm with BSIMM6

The most important use of the BSIMM is as a measuring stick to determine where your approach currently stands relative to other firms. Note which activities you already have in place, find them in the skeleton, then determine their levels and build a scorecard. A direct comparison of all 112 activities is perhaps the most obvious use of the BSIMM. This can be accomplished by building a scorecard using the data above.

The scorecard you see on the next page depicts a fake firm that performs 37 BSIMM activities (noted as 1s in the FIRM columns), including eight activities that are the most common in their respective practices (purple boxes). On the other hand, the firm does not perform the most commonly observed activities in the other four practices (red boxes) and should take some time to determine whether these are necessary or useful to its overall software security initiative. The BSIMM6 FIRM columns show the number of observations (currently out of 78) for each activity, allowing the firm to understand the general popularity of an activity among the 78 BSIMM firms.

BSIMM SCORECARD FOR: FIRM | OBSERVATIONS: 37

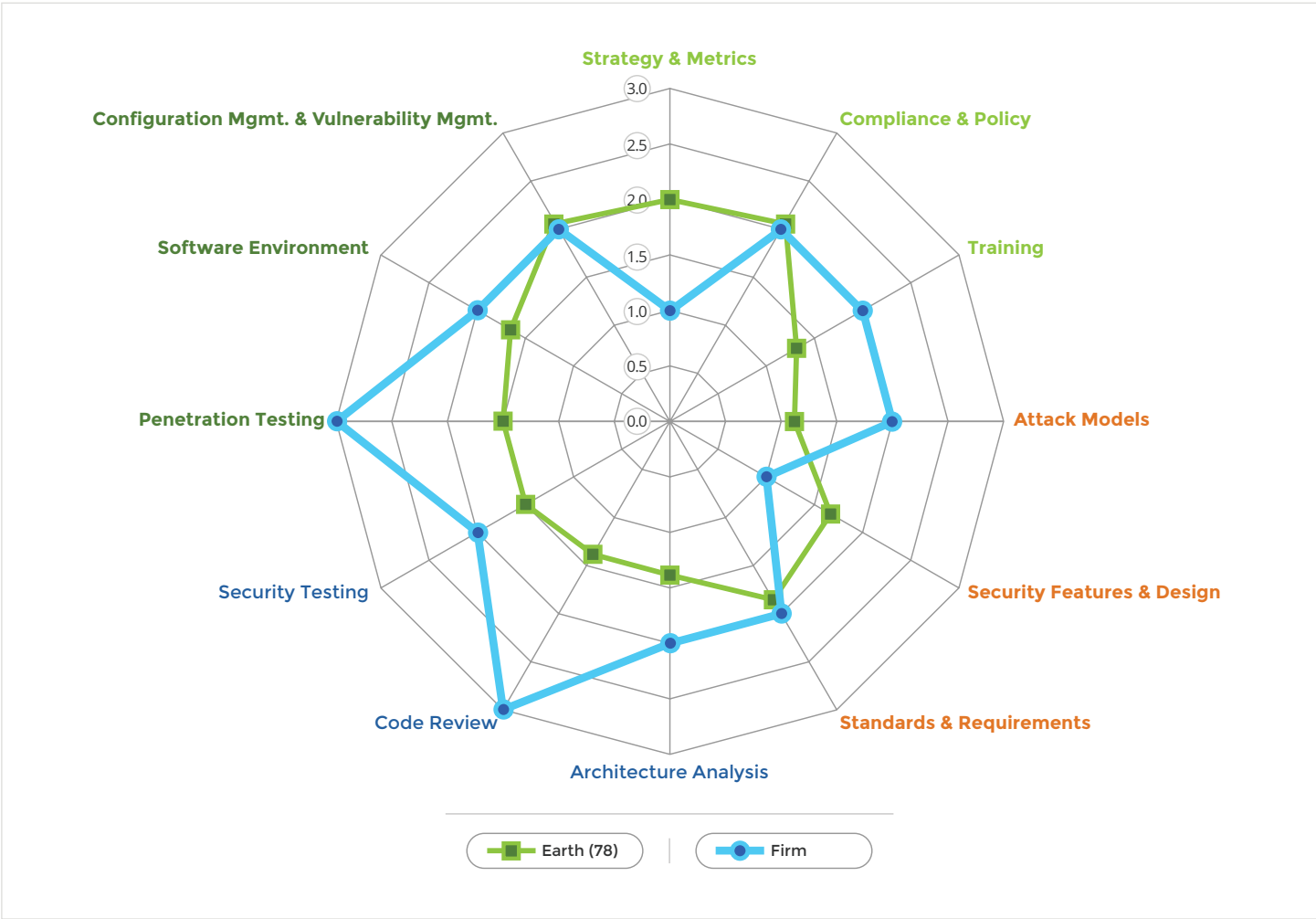
| GOVERNANCE | | | INTELLIGENCE | | | SSDL TOUCHPOINTS | | | DEPLOYMENT | | |
|---------------------|--------------|------|----------------------------|--------------|------|-----------------------|--------------|------|---------------------------|--------------|------|
| ACTIVITY | BSIMM6 FIRMS | FIRM | ACTIVITY | BSIMM6 FIRMS | FIRM | ACTIVITY | BSIMM6 FIRMS | FIRM | ACTIVITY | BSIMM6 FIRMS | FIRM |
| STRATEGY & METRICS | | | ATTACK MODELS | | | ARCHITECTURE ANALYSIS | | | PENETRATION TESTING | | |
| [SM1.1] | 41 | 1 | [AM1.1] | 17 | 1 | [AA1.1] | 67 | 1 | [PT1.1] | 69 | 1 |
| [SM1.2] | 40 | | [AM1.2] | 51 | | [AA1.2] | 29 | 1 | [PT1.2] | 47 | 1 |
| [SM1.3] | 36 | 1 | [AM1.3] | 31 | | [AA1.3] | 22 | 1 | [PT1.3] | 47 | |
| [SM1.4] | 66 | 1 | [AM1.4] | 8 | 1 | [AA1.4] | 46 | | [PT2.2] | 20 | 1 |
| [SM2.1] | 36 | | [AM1.5] | 46 | 1 | [AA2.1] | 12 | | [PT2.3] | 17 | |
| [SM2.2] | 29 | | [AM1.6] | 11 | | [AA2.2] | 9 | 1 | [PT3.1] | 10 | 1 |
| [SM2.3] | 30 | | [AM2.1] | 6 | | [AA2.3] | 13 | | [PT3.2] | 8 | |
| [SM2.5] | 17 | | [AM2.2] | 8 | 1 | [AA3.1] | 6 | | | | |
| [SM2.6] | 29 | | [AM3.1] | 4 | | [AA3.2] | 1 | | | | |
| [SM3.1] | 15 | | [AM3.2] | 2 | | | | | | | |
| [SM3.2] | 7 | | | | | | | | | | |
| COMPLIANCE & POLICY | | | SECURITY FEATURES & DESIGN | | | CODE REVIEW | | | SOFTWARE ENVIRONMENT | | |
| [CP1.1] | 45 | 1 | [SFD1.1] | 61 | | [CR1.1] | 18 | | [SE1.1] | 37 | |
| [CP1.2] | 61 | | [SFD1.2] | 59 | 1 | [CR1.2] | 53 | 1 | [SE1.2] | 69 | 1 |
| [CP1.3] | 41 | 1 | [SFD2.1] | 24 | | [CR1.4] | 55 | 1 | [SE2.2] | 31 | 1 |
| [CP2.1] | 19 | | [SFD2.2] | 39 | | [CR1.5] | 24 | | [SE2.4] | 25 | |
| [CP2.2] | 23 | | [SFD3.1] | 8 | | [CR1.6] | 27 | 1 | [SE3.2] | 10 | |
| [CP2.3] | 25 | | [SFD3.2] | 11 | | [CR2.2] | 7 | | [SE3.3] | 5 | |
| [CP2.4] | 29 | | [SFD3.3] | 2 | | [CR2.5] | 20 | | | | |
| [CP2.5] | 33 | 1 | | | | [CR2.6] | 16 | | | | |
| [CP3.1] | 18 | | | | | [CR3.2] | 3 | 1 | | | |
| [CP3.2] | 11 | | | | | [CR3.3] | 5 | | | | |
| [CP3.3] | 6 | | | | | [CR3.4] | 3 | | | | |
| TRAINING | | | STANDARDS & REQUIREMENTS | | | SECURITY TESTING | | | CONFIG. MGMT & VULN. MGMT | | |
| [T1.1] | 59 | 1 | [SR 1.1] | 57 | 1 | [ST1.1] | 61 | 1 | [CMVM1.1] | 71 | 1 |
| [T1.5] | 26 | | [SR1.2] | 50 | | [ST1.3] | 66 | 1 | [CMVM1.2] | 73 | |
| [T1.6] | 17 | 1 | [SR1.3] | 52 | 1 | [ST2.1] | 24 | 1 | [CMVM2.1] | 64 | 1 |
| [T1.7] | 36 | | [SR2.2] | 27 | | [ST2.4] | 8 | | [CMVM2.2] | 61 | |
| [T2.5] | 10 | | [SR2.3] | 21 | | [ST2.5] | 10 | | [CMVM2.3] | 31 | |
| [T2.6] | 15 | 1 | [SR2.4] | 19 | | [ST2.6] | 11 | | [CMVM3.1] | 4 | |
| [T2.7] | 6 | | [SR2.5] | 20 | 1 | [ST3.3] | 4 | | [CMVM3.2] | 6 | |
| [T3.1] | 3 | | [SR2.6] | 23 | 1 | [ST3.4] | 4 | | [CMVM3.3] | 6 | |
| [T3.2] | 3 | | [SR3.1] | 6 | | [ST3.5] | 5 | | [CMVM3.4] | 3 | |
| [T3.3] | 3 | | [SR3.2] | 11 | | | | | | | |
| [T3.4] | 8 | | | | | | | | | | |
| [T3.5] | 4 | | | | | | | | | | |

| | | |
|---------|--------------|---|
| LEGEND: | ACTIVITY | 112 BSIMM6 activities, shown in 4 domains and 12 practices |
| | BSIMM6 FIRMS | count of firms (out of 78) observed performing each activity |
| | | most common activity within a practice |
| | | most common activity not observed in this assessment |
| | 1 | most common activity was observed in this assessment |
| | | a practice where firm's high-water mark score is below the BSIMM6 average |

Once you have determined where you stand with activities, you can devise a plan to enhance practices with other activities included in the BSIMM. By providing actual measurement data from the field, the BSIMM makes it possible to build a long-term plan for a software security initiative and track progress against that plan. For the record, there’s no inherent reason to adopt all activities in every level for each practice. Adopt those activities that make sense for your organization and ignore those that don’t.

In our own work using the BSIMM to assess initiatives, we found that creating a spider chart yielding a “high-water mark” approach (based on the three levels per practice) is sufficient to get a low-resolution feel for maturity, especially when working with data from a particular vertical.

SPIDER CHART FOR FAKE FIRM



One meaningful comparison is to chart your own high-water mark against the averages we have published to see how your initiative stacks up. Above, we have plotted data from the fake firm against the BSIMM Earth (all participating firms) graph. The breakdown of activities into levels for each practice is meant only as a guide. The levels provide a natural progression through the activities associated with each practice. However, it’s not at all necessary to carry out all activities in a given level before moving on to activities at a higher level in the same practice. That said, the levels we have identified hold water under statistical scrutiny. Level 1 activities (straightforward and simple) are commonly observed, Level 2 (more difficult and requiring more coordination) slightly less so, and Level 3 (rocket science) are rarely observed.

By identifying activities from each practice that would work for you, and by ensuring proper balance with respect to domains, you can create a strategic plan for your software security initiative moving forward. Note that most software security initiatives are multi-year efforts with real budget, mandate, and ownership behind them. Though all initiatives look different and are tailored to fit a particular organization, as we describe on page 20, all initiatives do share common core activities.

BSIMM6 Analysis

The BSIMM has produced a wealth of real-world data about software security.

BSIMM Over Time

This is the sixth major release of the BSIMM and the chart below shows how the BSIMM has grown over the iterations. (Recall that our data freshness constraints, introduced with BSIMM-V and tightened for BSIMM6, cause data from firms with aging measurements to be removed from the data set.) BSIMM6 describes the work of 3,195 people working full-time in software security, directly impacting the security efforts of 287,006 developers.

We introduce two new verticals in BSIMM6: Consumer Electronics and Healthcare.

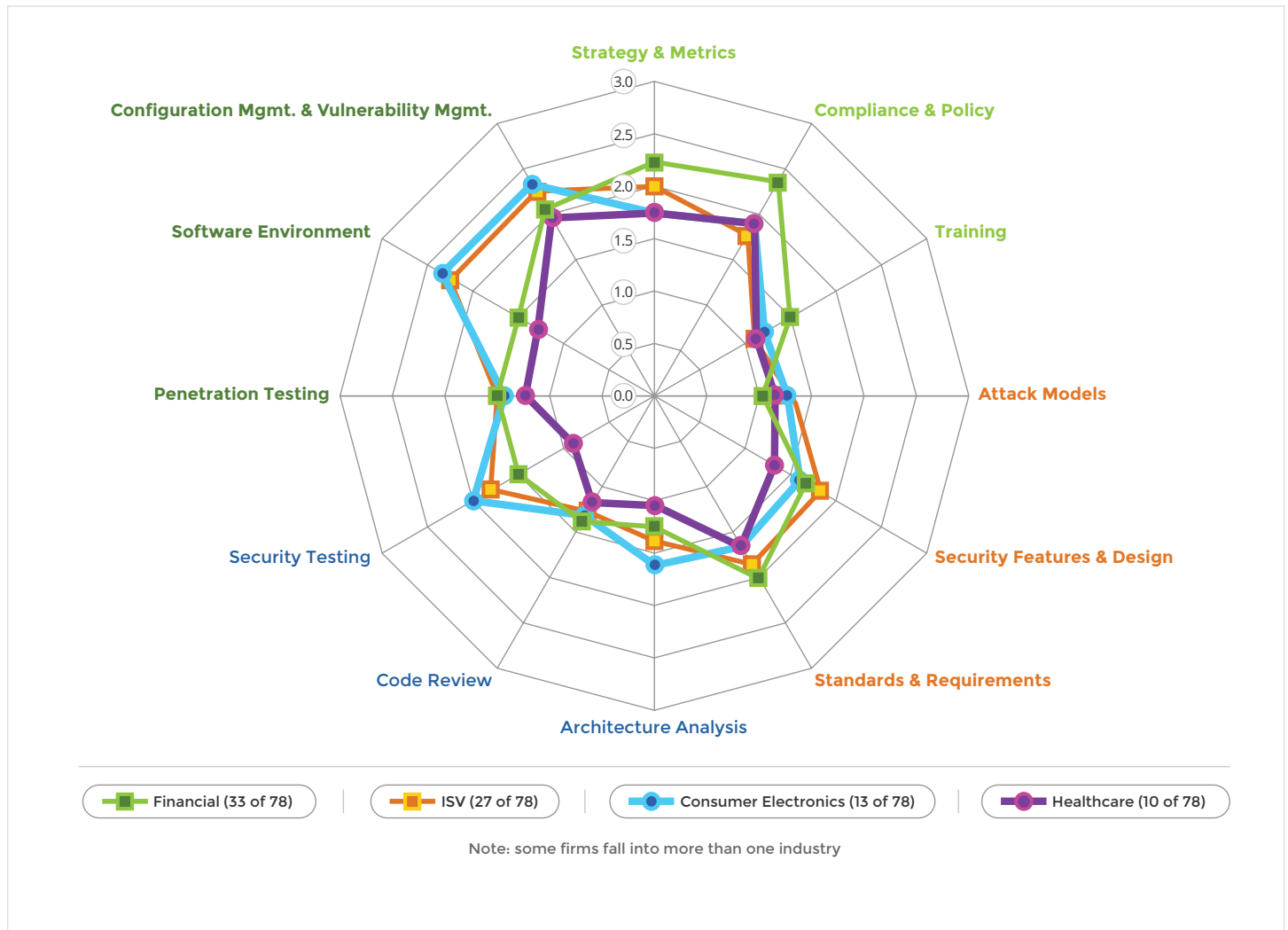
BSIMM NUMBERS OVER TIME

| | BSIMM6 | BSIMM-V | BSIMM4 | BSIMM3 | BSIMM2 | BSIMM1 |
|--------------------------|----------|---------|----------|----------|----------|----------|
| FIRMS | 78 | 67 | 51 | 42 | 30 | 9 |
| MEASUREMENTS | 202 | 161 | 95 | 81 | 49 | 9 |
| 2 ND MEASURES | 26 | 21 | 13 | 11 | 0 | 0 |
| 3 RD MEASURES | 10 | 4 | 1 | 0 | 0 | 0 |
| SSG MEMBERS | 1,084 | 976 | 978 | 786 | 635 | 370 |
| SATELLITE MEMBERS | 2,111 | 1,954 | 2,039 | 1,750 | 1,150 | 710 |
| DEVELOPERS | 287,006 | 272,358 | 218,286 | 185,316 | 141,175 | 67,950 |
| APPLICATIONS | 69,750 | 69,039 | 58,739 | 41,157 | 28,243 | 3,970 |
| AVG. SSG AGE (in yrs.) | 3.98 | 4.28 | 4.13 | 4.32 | 4.49 | 5.32 |
| SSG AVG. OF AVGS | 1.51/100 | 1.4/100 | 1.95/100 | 1.99/100 | 1.02/100 | 1.13/100 |
| FINANCIAL SERVICES | 33 | 26 | 19 | 17 | 12 | 4 |
| ISVs | 27 | 25 | 19 | 15 | 7 | 4 |
| HEALTHCARE | 10 | | | | | |
| CONSUMER ELECTRONICS | 13 | | | | | |

The BSIMM and Industry Verticals

The spider charts we introduced earlier are also useful for comparing groups of firms from particular industry verticals. The graph below shows data from four verticals charted together: financial services (33), independent software vendors (27), consumer electronics (13), and healthcare (10).

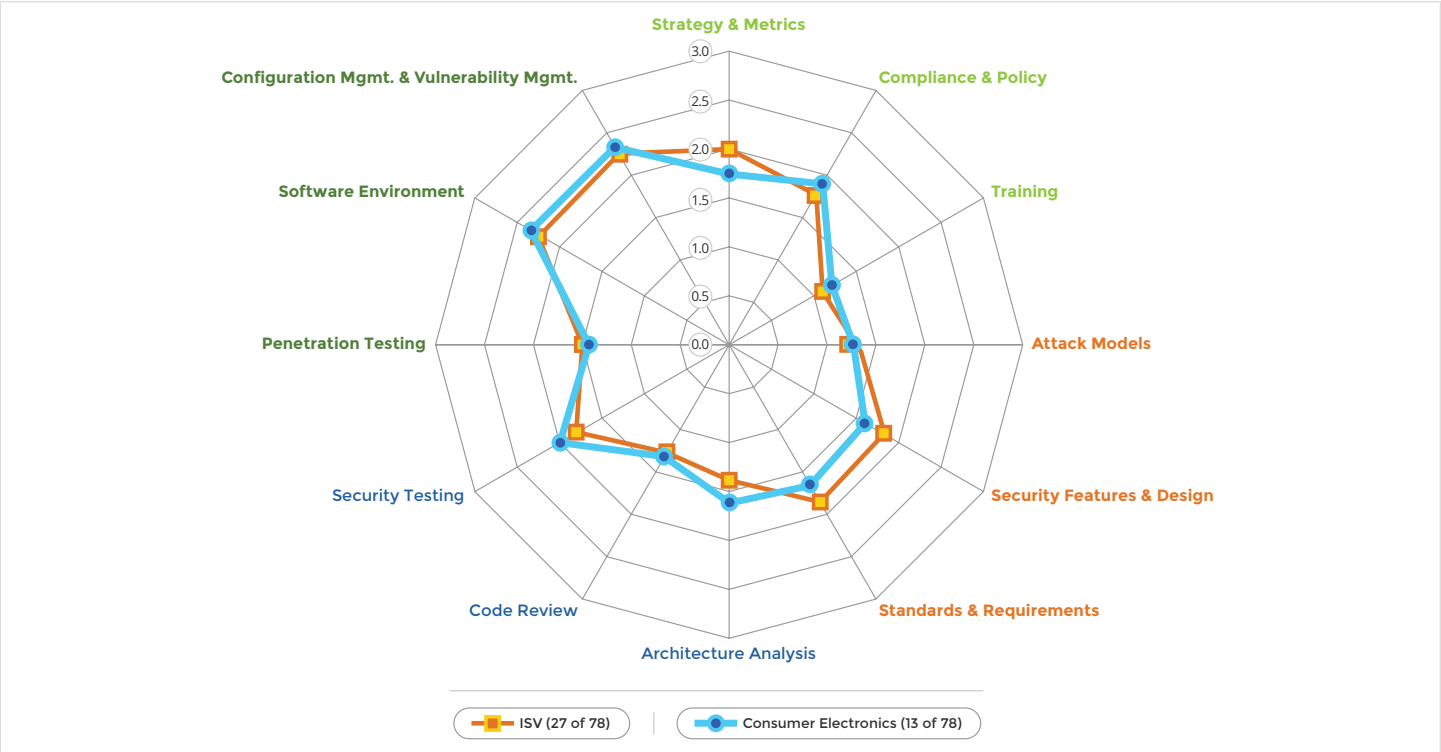
VERTICAL COMPARISON SPIDER CHART



When it comes to the two best-represented verticals, on average, financial services firms (FIs) have greater maturity as compared to independent software vendors (ISVs) in five of the 12 practices. By the same measure, independent software vendors have greater maturity as compared to financial services firms in seven of the 12 practices. Though there is plenty of overlap here, major differences can be explained with reference to the move to cloud computing among ISVs (see CMVM and SE) while FIs tend to emphasize standards and compliance practices (see SR and CP).

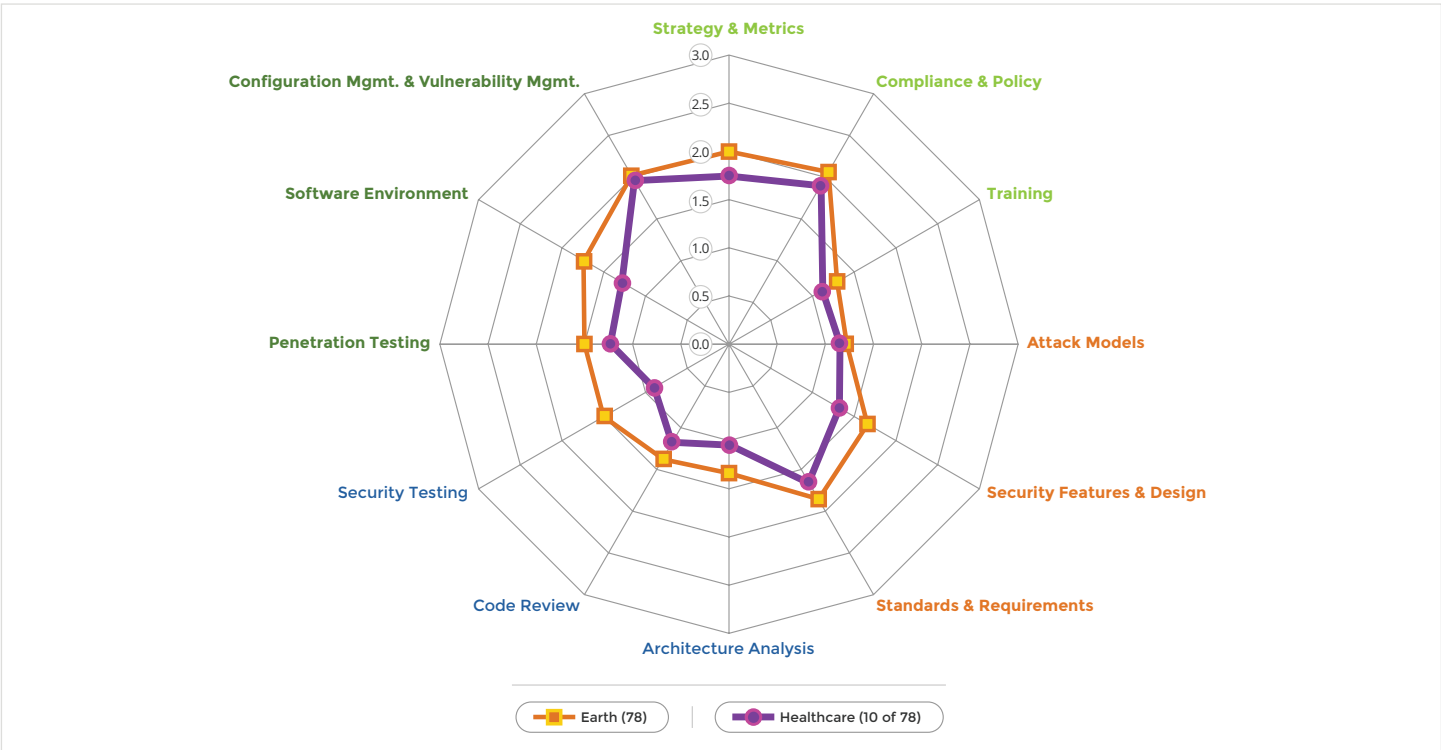
The consumer electronics vertical is very similar to the independent software vendor vertical as shown in the spider chart below. The security of modern consumer electronics relies heavily on the software included on the device. Many modern consumer devices are connected directly to the Internet and must negotiate the associated risks appropriately.

ISV vs. CE SPIDER CHART



The most interesting new vertical in BSIMM6 is healthcare, which is appreciably more nascent than the other three verticals. Simply put, healthcare firms are just getting started with software security. However, according to NH-ISAC (a healthcare industry security consortium), firms using BSIMM are further ahead than most of the other healthcare companies (many of which remain unaware of the need to address software security). Healthcare is likely to mature quickly now that software security has come into stark focus.

EARTH vs. HEALTHCARE SPIDER CHART



In the table below, you can see the BSIMM scorecards for the four verticals compared side by side. In the Activity columns, we have highlighted in yellow the most common activity in each practice as observed in the entire BSIMM data pool (78 firms).

SCORECARD COMPARING INDUSTRY VERTICALS

| GOVERNANCE | | | | |
|------------|----------------------|----------------|-----------------------|---------------|
| ACTIVITY | FINANCIAL (of 33) | ISV (of 27) | HEALTHCARE (of 10) | CE (of 13) |
| [SM1.1] | 21 | 15 | 4 | 7 |
| [SM1.2] | 16 | 16 | 5 | 8 |
| [SM1.3] | 17 | 12 | 3 | 6 |
| [SM1.4] | 30 | 24 | 7 | 9 |
| [SM2.1] | 19 | 13 | 3 | 4 |
| [SM2.2] | 14 | 10 | 3 | 3 |
| [SM2.3] | 10 | 14 | 5 | 5 |
| [SM2.5] | 12 | 2 | 1 | 2 |
| [SM2.6] | 17 | 9 | 2 | 4 |
| [SM3.1] | 10 | 3 | 1 | 1 |
| [SM3.2] | 0 | 4 | 0 | 3 |
| | | | | |
| [CP1.1] | 22 | 13 | 8 | 8 |
| [CP1.2] | 25 | 19 | 10 | 12 |
| [CP1.3] | 23 | 11 | 6 | 6 |
| [CP2.1] | 12 | 5 | 1 | 1 |
| [CP2.2] | 11 | 8 | 3 | 3 |
| [CP2.3] | 10 | 12 | 4 | 6 |
| [CP2.4] | 16 | 8 | 3 | 4 |
| [CP2.5] | 14 | 9 | 5 | 4 |
| [CP3.1] | 16 | 0 | 1 | 0 |
| [CP3.2] | 4 | 1 | 2 | 0 |
| [CP3.3] | 3 | 2 | 0 | 1 |
| | | | | |
| [T1.1] | 27 | 19 | 8 | 10 |
| [T1.5] | 15 | 9 | 1 | 3 |
| [T1.6] | 6 | 8 | 1 | 5 |
| [T1.7] | 20 | 13 | 3 | 4 |
| [T2.5] | 3 | 3 | 2 | 3 |
| [T2.6] | 8 | 3 | 2 | 2 |
| [T2.7] | 1 | 3 | 0 | 2 |
| [T3.1] | 1 | 2 | 0 | 0 |
| [T3.2] | 1 | 1 | 1 | 0 |
| [T3.3] | 0 | 2 | 0 | 1 |
| [T3.4] | 5 | 2 | 0 | 1 |
| [T3.5] | 1 | 0 | 1 | 1 |

| INTELLIGENCE | | | | |
|--------------|----------------------|----------------|-----------------------|---------------|
| ACTIVITY | FINANCIAL (of 33) | ISV (of 27) | HEALTHCARE (of 10) | CE (of 13) |
| [AM1.1] | 7 | 7 | 1 | 2 |
| [AM1.2] | 27 | 14 | 6 | 8 |
| [AM1.3] | 16 | 10 | 3 | 5 |
| [AM1.4] | 1 | 3 | 1 | 2 |
| [AM1.5] | 23 | 15 | 4 | 8 |
| [AM1.6] | 1 | 7 | 0 | 3 |
| [AM2.1] | 2 | 3 | 2 | 0 |
| [AM2.2] | 1 | 4 | 0 | 2 |
| [AM3.1] | 0 | 4 | 1 | 1 |
| [AM3.2] | 0 | 2 | 0 | 0 |
| | | | | |
| [SFD1.1] | 29 | 21 | 7 | 10 |
| [SFD1.2] | 27 | 23 | 6 | 11 |
| [SFD2.1] | 11 | 9 | 0 | 5 |
| [SFD2.2] | 15 | 16 | 3 | 8 |
| [SFD3.1] | 5 | 1 | 0 | 1 |
| [SFD3.2] | 5 | 5 | 1 | 0 |
| [SFD3.3] | 0 | 2 | 0 | 0 |
| | | | | |
| [SR1.1] | 29 | 22 | 5 | 9 |
| [SR1.2] | 23 | 15 | 9 | 7 |
| [SR1.3] | 21 | 19 | 7 | 11 |
| [SR2.2] | 15 | 9 | 3 | 3 |
| [SR2.3] | 11 | 5 | 3 | 2 |
| [SR2.4] | 5 | 9 | 2 | 4 |
| [SR2.5] | 11 | 4 | 2 | 4 |
| [SR2.6] | 12 | 8 | 0 | 4 |
| [SR3.1] | 2 | 3 | 0 | 2 |
| [SR3.2] | 6 | 3 | 2 | 2 |

| SSDL TOUCHPOINTS | | | | |
|------------------|----------------------|----------------|-----------------------|---------------|
| ACTIVITY | FINANCIAL (of 33) | ISV (of 27) | HEALTHCARE (of 10) | CE (of 13) |
| [AA1.1] | 29 | 24 | 7 | 11 |
| [AA1.2] | 8 | 14 | 2 | 7 |
| [AA1.3] | 7 | 11 | 1 | 5 |
| [AA1.4] | 25 | 10 | 7 | 6 |
| [AA2.1] | 5 | 4 | 2 | 3 |
| [AA2.2] | 1 | 4 | 1 | 4 |
| [AA2.3] | 4 | 6 | 0 | 3 |
| [AA3.1] | 2 | 1 | 0 | 2 |
| [AA3.2] | 0 | 0 | 0 | 1 |
| | | | | |
| [CR1.1] | 9 | 5 | 0 | 3 |
| [CR1.2] | 26 | 17 | 6 | 7 |
| [CR1.4] | 21 | 19 | 9 | 7 |
| [CR1.5] | 7 | 12 | 3 | 4 |
| [CR1.6] | 13 | 8 | 3 | 4 |
| [CR2.2] | 2 | 3 | 0 | 2 |
| [CR2.5] | 10 | 5 | 3 | 4 |
| [CR2.6] | 8 | 5 | 0 | 3 |
| [CR3.2] | 2 | 1 | 0 | 0 |
| [CR3.3] | 2 | 1 | 0 | 1 |
| [CR3.4] | 3 | 0 | 0 | 0 |
| | | | | |
| [ST1.1] | 28 | 21 | 7 | 10 |
| [ST1.3] | 28 | 27 | 6 | 12 |
| [ST2.1] | 13 | 10 | 1 | 3 |
| [ST2.4] | 4 | 2 | 0 | 2 |
| [ST2.5] | 3 | 4 | 0 | 3 |
| [ST2.6] | 0 | 9 | 0 | 6 |
| [ST3.3] | 1 | 2 | 0 | 1 |
| [ST3.4] | 0 | 2 | 0 | 4 |
| [ST3.5] | 1 | 3 | 0 | 2 |

| DEPLOYMENT | | | | |
|------------|----------------------|----------------|-----------------------|---------------|
| ACTIVITY | FINANCIAL (of 33) | ISV (of 27) | HEALTHCARE (of 10) | CE (of 13) |
| [PT1.1] | 31 | 22 | 9 | 11 |
| [PT1.2] | 23 | 17 | 3 | 6 |
| [PT1.3] | 18 | 15 | 7 | 8 |
| [PT2.2] | 3 | 10 | 2 | 4 |
| [PT2.3] | 6 | 8 | 1 | 2 |
| [PT3.1] | 1 | 4 | 1 | 4 |
| [PT3.2] | 3 | 3 | 0 | 2 |
| | | | | |
| [SE1.1] | 18 | 13 | 5 | 4 |
| [SE1.2] | 31 | 22 | 9 | 10 |
| [SE2.2] | 10 | 17 | 0 | 9 |
| [SE2.4] | 5 | 16 | 1 | 9 |
| [SE3.2] | 0 | 8 | 1 | 5 |
| [SE3.3] | 1 | 3 | 0 | 1 |
| | | | | |
| [CMVM1.1] | 31 | 25 | 8 | 13 |
| [CMVM1.2] | 31 | 26 | 9 | 12 |
| [CMVM2.1] | 30 | 23 | 7 | 10 |
| [CMVM2.2] | 25 | 24 | 6 | 11 |
| [CMVM2.3] | 13 | 11 | 6 | 5 |
| [CMVM3.1] | 0 | 1 | 0 | 3 |
| [CMVM3.2] | 2 | 2 | 0 | 2 |
| [CMVM3.3] | 3 | 3 | 1 | 0 |
| [CMVM3.4] | 0 | 2 | 0 | 0 |

BSIMM as a Longitudinal Study

Twenty-six of the 78 firms in BSIMM6 have been measured twice. On average, the time between the two measurements was 24.8 months. Though individual activities among the 12 practices come and go (as shown in the Longitudinal Scorecard on the next page), in general, re-measurement over time shows a clear trend of increased maturity in the population of the 26 firms re-measured thus far. The raw score went up in 21 of the 26 firms. Across all 26 firms, the observation count increased by an average of 10 (29.6%). Software security initiatives mature over time.

Software security initiatives mature over time.

LONGITUDINAL SCORECARD AS MEASURED OVER TIME

| GOVERNANCE | | | INTELLIGENCE | | | SSDL TOUCHPOINTS | | | DEPLOYMENT | | |
|------------|-----------------------|-----------------------|--------------|-----------------------|-----------------------|------------------|-----------------------|-----------------------|------------|-----------------------|-----------------------|
| ACTIVITY | BSIMM ROUND 1 (OF 26) | BSIMM ROUND 2 (OF 26) | ACTIVITY | BSIMM ROUND 1 (OF 26) | BSIMM ROUND 2 (OF 26) | ACTIVITY | BSIMM ROUND 1 (OF 26) | BSIMM ROUND 2 (OF 26) | ACTIVITY | BSIMM ROUND 1 (OF 26) | BSIMM ROUND 2 (OF 26) |
| [SM1.1] | 16 | 24 | [AM1.1] | 7 | 9 | [AA1.1] | 20 | 25 | [PT1.1] | 25 | 25 |
| [SM1.2] | 16 | 16 | [AM1.2] | 17 | 20 | [AA1.2] | 16 | 17 | [PT1.2] | 17 | 22 |
| [SM1.3] | 15 | 18 | [AM1.3] | 10 | 14 | [AA1.3] | 13 | 16 | [PT1.3] | 15 | 16 |
| [SM1.4] | 20 | 25 | [AM1.4] | 8 | 6 | [AA1.4] | 16 | 18 | [PT2.2] | 10 | 8 |
| [SM2.1] | 11 | 17 | [AM1.5] | 17 | 21 | [AA2.1] | 7 | 9 | [PT2.3] | 12 | 11 |
| [SM2.2] | 9 | 13 | [AM1.6] | 5 | 9 | [AA2.2] | 3 | 4 | [PT3.1] | 5 | 6 |
| [SM2.3] | 15 | 12 | [AM2.1] | 7 | 6 | [AA2.3] | 9 | 8 | [PT3.2] | 4 | 4 |
| [SM2.5] | 7 | 12 | [AM2.2] | 6 | 8 | [AA3.1] | 6 | 7 | | | |
| [SM2.6] | 15 | 17 | [AM3.1] | 1 | 2 | [AA3.2] | 1 | 1 | | | |
| [SM3.1] | 5 | 11 | [AM3.2] | 0 | 3 | | | | | | |
| [SM3.2] | 2 | 3 | | | | | | | | | |
| [CP1.1] | 18 | 18 | [SFD1.1] | 23 | 20 | [CR1.1] | 6 | 11 | [SE1.1] | 10 | 12 |
| [CP1.2] | 23 | 23 | [SFD1.2] | 15 | 23 | [CR1.2] | 12 | 19 | [SE1.2] | 24 | 23 |
| [CP1.3] | 20 | 23 | [SFD2.1] | 10 | 12 | [CR1.4] | 16 | 22 | [SE2.2] | 11 | 12 |
| [CP2.1] | 11 | 10 | [SFD2.2] | 9 | 18 | [CR1.5] | 9 | 12 | [SE2.4] | 9 | 15 |
| [CP2.2] | 13 | 11 | [SFD3.1] | 6 | 7 | [CR1.6] | 10 | 16 | [SE3.2] | 4 | 6 |
| [CP2.3] | 14 | 10 | [SFD3.2] | 5 | 8 | [CR2.2] | 7 | 6 | [SE3.3] | 6 | 2 |
| [CP2.4] | 8 | 15 | [SFD3.3] | 4 | 2 | [CR2.5] | 9 | 12 | | | |
| [CP2.5] | 13 | 17 | | | | [CR2.6] | 5 | 12 | | | |
| [CP3.1] | 5 | 10 | | | | [CR3.2] | 1 | 1 | | | |
| [CP3.2] | 5 | 8 | | | | [CR3.3] | 1 | 3 | | | |
| [CP3.3] | 2 | 7 | | | | [CR3.4] | 0 | 0 | | | |
| [T1.1] | 18 | 25 | [SR1.1] | 20 | 22 | [ST1.1] | 18 | 19 | [CMVM1.1] | 22 | 26 |
| [T1.5] | 8 | 17 | [SR1.2] | 18 | 23 | [ST1.3] | 18 | 22 | [CMVM1.2] | 22 | 24 |
| [T1.6] | 10 | 12 | [SR1.3] | 14 | 22 | [ST2.1] | 13 | 13 | [CMVM2.1] | 24 | 24 |
| [T1.7] | 11 | 21 | [SR2.2] | 14 | 17 | [ST2.4] | 7 | 7 | [CMVM2.2] | 15 | 23 |
| [T2.5] | 5 | 6 | [SR2.3] | 9 | 10 | [ST2.5] | 4 | 5 | [CMVM2.3] | 15 | 14 |
| [T2.6] | 3 | 8 | [SR2.4] | 9 | 10 | [ST2.6] | 7 | 7 | [CMVM3.1] | 2 | 2 |
| [T2.7] | 7 | 7 | [SR2.5] | 7 | 13 | [ST3.3] | 3 | 4 | [CMVM3.2] | 3 | 5 |
| [T3.1] | 3 | 4 | [SR2.6] | 11 | 12 | [ST3.4] | 2 | 2 | [CMVM3.3] | 0 | 0 |
| [T3.2] | 0 | 4 | [SR3.1] | 5 | 3 | [ST3.5] | 4 | 6 | [CMVM3.4] | 0 | 1 |
| [T3.3] | 1 | 3 | [SR3.2] | 5 | 6 | | | | | | |
| [T3.4] | 0 | 6 | | | | | | | | | |
| [T3.5] | 2 | 6 | | | | | | | | | |

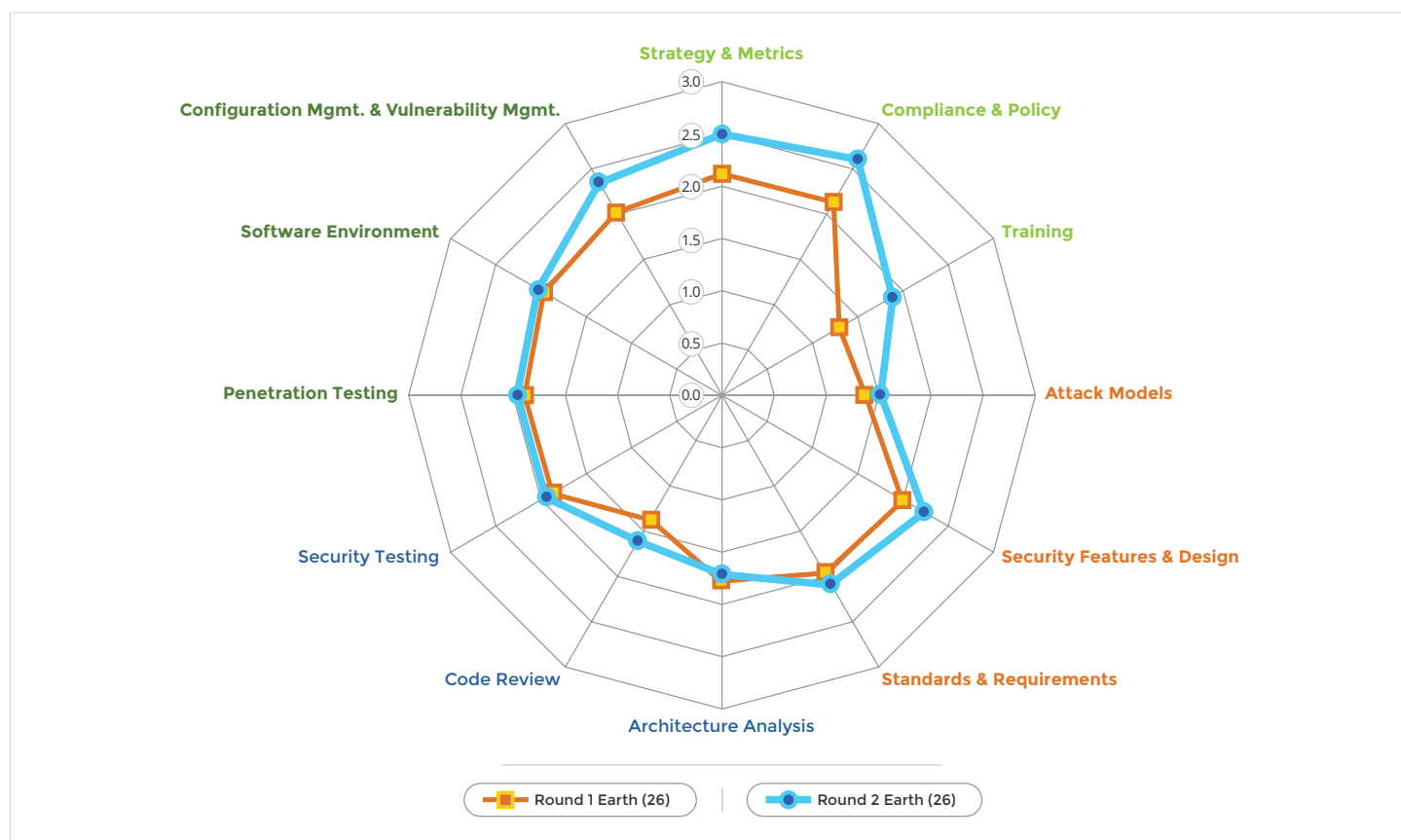
Here are two ways of thinking about the change represented by the Longitudinal Scorecard (showing 26 firms moving from their first to their second assessment). We see the biggest changes in these activities: [CR1.2 Have SSG perform ad hoc reviews] and [T1.7 Offer on-demand individual training], with 11 new observations; and [SM2.1 Publish data about software security internally], [T1.5 Offer role-specific advanced curriculum], [SR1.3 Translate compliance constraints to requirements], and [SFD2.2 Create SSG capability to solve difficult design problems], each with 10 new observations.

However, this cannot be seen directly on the Longitudinal Scorecard. For example, [CR1.2 Have SSG perform ad hoc

reviews] was a new activity for 11 firms, but the scorecard clearly shows that the total observations increased by only seven. What happened? There was “churn.” While the activity was newly observed in 11 firms, it was no longer observed in four firms, giving a total change of seven (as shown in the scorecard).

As a different example, the activity [SM1.2 Create evangelism role to perform internal marketing] was both newly observed in six firms and no longer observed in six firms. Therefore, total observation count remains unchanged on the scorecard above. Oddly enough, the same type of zero-sum “churn” also occurred in 14 other activities.

ROUND 1 EARTH vs. ROUND 2 EARTH



BSIMM Community

The 78 firms participating in the BSIMM make up the BSIMM Community. A moderated private mailing list with more than 250 members allows SSG leaders participating in the BSIMM to discuss solutions with others who face the same issues, discuss strategy with someone who has already addressed an issue, seek out mentors from those further along a career path, and band together to solve hard problems.

The BSIMM Community also hosts annual private conferences where representatives from each firm gather together in an off-the-record forum to discuss software security initiatives. To date, nine BSIMM Community conferences have been held, five in the United States and four in Europe. During the conferences, representatives from BSIMM firms give the presentations.

The [BSIMM website](https://www.BSIMM.com) (https://www.BSIMM.com) includes a credentialed BSIMM Community section where information from the conferences, working groups and mailing list-initiated studies are posted.

PART TWO

This part of the document provides detail behind the people and activities involved in a software security initiative and measured by the BSIMM. We begin by describing the software security group (SSG) and other key roles. We then provide descriptions for each of the 112 activities that comprise BSIMM6. Throughout, we annotate our discussion with relevant observation data from the 78 participating firms.

Roles in a Software Security Initiative

Determining who is supposed to carry out the activities described in the BSIMM is an important part of making any software security initiative work.

Executive Leadership

Of primary interest is identifying and empowering a senior executive to manage operations, garner resources, and provide political cover for a software security initiative. Grassroots approaches to software security sparked and led solely by developers and their direct managers have a poor track record in the real world. Likewise, initiatives spearheaded by resources from an existing network security group often run into serious trouble when it comes time to interface with development groups. By identifying a senior executive and putting him or her in charge of software security directly, you address two management 101 concerns—accountability and empowerment. You also create a place in the organization where software security can take root and begin to thrive.

The executives in charge of the software security initiatives we studied have a variety of titles, including: Information Security Manager, Security Director, Director of Application Security Strategy, Director of Enterprise Information Security, VP of Security Architecture, Chief Product Security and Privacy Officer, CISO, Head of Information Governance, Global Head of Security Maturation and Certification, Technical Director, Director of IT Security, and Manager of Enterprise Information Protection. We observed a fairly wide spread in exactly where the SSG is situated in the firms we studied. In particular, 28 exist in the CIO's organization, 16 exist in the CTO's organization, 10 report to the COO, four report to the Chief Assurance Officer, two report to the CSO, and one SSG reports to each of the General Counsel, the CFO, and the founder. Twelve SSGs report up through technology or product operations groups (as opposed to governance organizations). Three report up through the business unit where they were originally formed. Several companies we studied did not specify where their SSG fits in the larger organization.

Carrying out the activities in the BSIMM successfully without an SSG is very unlikely.

Software Security Group (SSG)

The second most important role in a software security initiative after the senior executive is that of the Software Security Group. Every single one of the 78 initiatives we describe in BSIMM6 has an SSG. Carrying out the activities in the BSIMM successfully without an SSG is very unlikely (and has never been observed in the field to date), so create an SSG before you start working to adopt the BSIMM activities. The best SSG members are software security people, but software security people are often impossible to find. If you must create software security types from scratch, start with developers and teach them about security. Starting with network security people and attempting to teach them about software, compilers, SDLCs, bug tracking, and everything else in the software universe usually fails to produce the desired results. Unfortunately, no amount of traditional security knowledge can overcome a lack of experience building software.

SSGs come in a variety of shapes and sizes. All good SSGs appear to include both people with deep coding experience and people with architectural chops. As you will see below, software security can't only be about finding specific bugs such as the [OWASP Top Ten](#). Code review is a very important best practice and to perform code review you must actually understand code (not to mention the huge piles of security bugs). However, the best code reviewers sometimes make very poor software architects and asking them to perform an architecture risk analysis will only result in blank stares. Make sure you cover architectural capabilities in your SSG as well as you do code. Finally, the SSG is often asked to mentor, train, and work directly with hundreds of developers. Communications skills, teaching capability, and good consulting horse sense are must-haves for at least a portion of the SSG staff. For more about this issue, see our SearchSecurity article based on SSG structure data gathered at the 2014 BSIMM Community Conference: [How to Build a Team for Software Security Management](#).

Though no two of the 78 firms we examined had exactly the same SSG structure (suggesting that there is no one set way to structure an SSG), we did observe some commonalities that are worth mentioning. At the highest level of organization, SSGs come in five major flavors: those that are 1) organized to provide software security services, 2) organized around setting policy, 3) mirroring business unit organizations, 4) organized with a hybrid policy and services approach, and 5) structured around managing a distributed network of others doing software security work. Some SSGs are highly distributed across a firm and others are very centralized. If we look across all of the SSGs in our study, there are several common "subgroups" that are often observed: people dedicated to policy, strategy, and metrics; internal "services" groups that (often separately) cover tools, penetration testing, and middleware development plus shepherding; incident response groups; groups responsible for training development and delivery; externally-facing marketing and communications groups; and vendor-control groups.

In the statistics reported above, we noted an average ratio of SSG to development of 1.51% across the entire group of 78 organizations we studied. That means one SSG member for every 75 developers when we average the ratios for each participating firm. The SSG with the largest ratio was 16.7% and the smallest was 0.03%. To remind you of the particulars in terms of actual bodies, SSG size on average among the 78 firms was 13.9 people (smallest 1, largest 130, median 6).

Satellite

In addition to the SSG, many software security programs have identified a number of individuals (often developers, testers, and architects) who share a basic interest in software security, but are not directly employed in the SSG. When people like this carry out software security activities, we call this group the *satellite*.

Sometimes the satellite is widely distributed, with one or two members in each product group. Sometimes the

satellite is more focused and gets together regularly to compare notes, learn new technologies, and expand the understanding of software security in an organization. Identifying and fostering a strong satellite is important to the success of many software security initiatives (but not all of them). Some BSIMM activities target the satellite explicitly.

Of particular interest, all 10 firms with the highest BSIMM scores have a satellite (100%), with an average satellite size of 131 people. Outside of the top 10, 30 of the remaining 68 firms have a satellite (44.1%). Of the 10 firms with the lowest BSIMM scores, none have a satellite. This suggests that as a software security initiative matures, its activities become distributed and institutionalized into the organizational structure. Among our population of 78 firms, initiatives tend to evolve from centralized and specialized in the beginning to decentralized and distributed (with an SSG at the core orchestrating things).

Everybody Else

Our survey participants have engaged everyone involved in the software development lifecycle as a means of addressing software security.

All 10 firms with the highest BSIMM scores have a satellite.

- **Builders**, including developers, architects, and their managers must practice security engineering, ensuring that the systems that we build are defensible and not riddled with holes. The SSG will interact directly with builders when they carry out the activities described in the BSIMM. Generally speaking, as an organization matures, the SSG attempts to empower builders so they can carry out most of the BSIMM activities themselves with the SSG helping in special cases and providing oversight. In this version of the BSIMM, we often don't explicitly point out whether a given activity is to be carried out by the SSG, developers, or testers. You should come up with an approach that makes sense for your organization and accounts for your workload and your software lifecycle.
- **Testers** concerned with routine testing and verification should do what they can to keep a weather eye out for security problems. Some of the BSIMM activities in the Security Testing practice (see page 52) can be carried out directly by QA.
- **Operations** people must continue to design reasonable networks, defend them, and keep them up. As you will see in the Deployment domain of the SSF, software security doesn't end when software is "shipped."
- **Administrators** must understand the distributed nature of modern systems and begin to practice the principle of least privilege, especially when it comes to the applications they host or attach to as services in the cloud.
- **Executives and middle management**, including line of business owners and product managers, must understand how early investment in security design and security analysis affects the degree to which users will trust their products. Business requirements should explicitly address security needs. Any sizeable business today depends on software to work. Software security is a business necessity.
- **Vendors**, including those who supply COTS, custom software and software-as-a-service, are increasingly subjected to SLAs and reviews (such as [vBSIMM](#)) that help ensure products are the result of a secure SDLC.

BSIMM6 Activities

We present a series of activities associated with each of the 12 BSIMM practices. Take a look at the BSIMM Skeleton in [Part One](#) if you need help understanding how the activities are organized. For each activity, we show the designator (using the example SM1.1, we see that SM1.1 is an activity in the Strategy & Metrics Level One, followed by the number of firms in the BSIMM performing that activity).



GOVERNANCE: Strategy & Metrics (SM)

The Strategy & Metrics practice encompasses planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and gates.

.....

SM LEVEL 1:

[SM1.1: 41] Publish process (roles, responsibilities, plan), evolve as necessary.

The process for addressing software security is broadcast to all stakeholders so that everyone knows the plan. Goals, roles, responsibilities, and activities are explicitly defined. Most organizations pick and choose from a published methodology such as the Microsoft SDL or the Cigital Touchpoints and then tailor the methodology to their needs. An SSDL process evolves as the organization matures and as the security landscape changes. A process must be published to count. In many cases, the methodology is published only internally and is controlled by the SSG. The SSDL does not need to be publicly promoted outside of the firm to have the desired impact.

[SM1.2: 40] Create evangelism role and perform internal marketing.

In order to build support for software security throughout the organization, someone in the SSG plays an evangelism role. This internal marketing function helps keep executives and all other stakeholders current on the magnitude of the software security problem and the elements of its solution. Evangelists might give talks for internal groups including executives, extend invitations to outside speakers, author white papers for internal consumption, or create a collection of papers, books, and other resources on an internal website and promote its use. Ad hoc conversations between the SSG and executives, or an SSG where “everyone is an evangelist,” do not achieve the desired results. A canonical example of such an evangelist was Michael Howard’s role at Microsoft just after the Gates memo.

[SM1.3: 36] Educate executives.

Executives are periodically shown the consequences of inadequate software security and the negative business impact that poor security can have. They’re also shown what other organizations are doing to attain software security. By understanding both the problem and its proper resolution, executives come to support the software security initiative as a risk management necessity. In its most dangerous form, such education arrives courtesy of malicious hackers or public data exposure incidents. Preferably, the SSG demonstrates a worst-case scenario in a controlled environment with the permission of all involved (e.g., actually showing working exploits and their business impact). In some cases, presentation to the Board can help garner resources for an ongoing software security initiative. Bringing in an outside guru is often helpful when seeking to bolster executive attention.

[SM1.4: 66] Identify gate locations, gather necessary artifacts.

The software security process includes release gates/checkpoints/milestones at one or more points in the SDLC or, more likely, the SDLCS. The first two steps toward establishing security-specific release gates are: 1) to identify gate

locations that are compatible with existing development practices and 2) to begin gathering the input necessary for making a go/no-go decision. Importantly at this stage, the gates are not enforced. For example, the SSG can collect security testing results for each project prior to release, but stop short of passing judgment on what constitutes sufficient testing or acceptable test results. The idea of identifying gates first and only enforcing them later is extremely helpful in moving development toward software security without major pain. Socialize the gates, and only turn them on once most projects already know how to succeed. This gradual approach serves to motivate good behavior without requiring it.

.....

SM LEVEL 2

[SM2.1: 36] Publish data about software security internally.

The SSG publishes data internally on the state of software security within the organization to facilitate improvement. The information might come as a dashboard with metrics for executives and software development management. Sometimes, publication is not shared with everyone in a firm, but rather with the relevant executives only. In this case, publishing information up to executives who then drive change in the organization is necessary. In other cases, open book management and publishing data to all stakeholders helps everyone know what's going on, with the philosophy that sunlight is the best disinfectant. If the organization's culture promotes internal competition between groups, this information adds a security dimension to the game.

[SM2.2: 29] Enforce gates with measurements and track exceptions.

SDLC security gates are now enforced: in order to pass a gate, a project must either meet an established measure or obtain a waiver. Even recalcitrant project teams must now play along. The SSG tracks exceptions. A gate could require a project to undergo code review and remediate any critical findings before release. In some cases, gates are directly associated with controls required by regulations, contractual agreements, and other business obligations, and exceptions are tracked as required by statutory or regulatory drivers. In other cases, gate measures yield key performance indicators that are used to govern the process. A revolving door or a rubber stamp exception process does not count. If some projects are automatically passed, that defeats the purpose of enforcing gates.

[SM2.3: 30] Create or grow a satellite.

The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill. Identifying this group is a step towards creating a social network that speeds the adoption of security into software development. One way to begin is to track the people who stand out during introductory training courses (see [T2.7 Identify satellite through training]).

Another way is to ask for volunteers. In a more top-down approach, initial satellite membership is assigned to ensure complete coverage of all development/product groups. Ongoing membership should be based on actual performance. A strong satellite is a good sign of a mature software security initiative.

**A strong satellite
is a good sign of a
mature software
security initiative.**

[SM2.5: 17] Identify metrics and use them to drive budgets.

The SSG and its management choose the metrics that define and measure software security initiative progress. These metrics will drive the initiative's budget and allocation of resources, so simple counts and statistics won't suffice. Metrics also allow the SSG to explain its goals and its progress in quantitative terms. One such metric could be security defect density. A reduction in security defect density could be used to show a decreasing cost of

.....

remediation over time. The key here is to tie technical results to business objectives in a clear and obvious fashion in order to justify funding. Because the concept of security is already tenuous to business people, making this explicit tie can be very helpful.

[SM2.6: 29] Require security sign-off.

The organization has an initiative-wide process for accepting security risk and documenting accountability. A risk acceptor signs off on the state of all software prior to release. For example, the sign-off policy might require the head of the business unit to sign off on critical vulnerabilities that have not been mitigated or SSDL steps that have been skipped. Informal risk acceptance alone does not count as security sign off, as the act of accepting risk is more effective when it is formalized (e.g., with a signature, form submission, or something similar) and captured for future reference. Similarly, simply stating that certain projects never need a sign-off does not achieve the desired results.

.....

SM Level 3

[SM3.1: 15] Use an internal tracking application with portfolio view.

The SSG uses a centralized tracking application to chart the progress of every piece of software in its purview. The application records the security activities scheduled, in progress and completed. It incorporates results from activities such as architecture analysis, code review, and security testing. The SSG uses the tracking application to generate portfolio reports for many of the metrics it uses. A combined inventory and risk posture view is fundamental. In many cases, these data are published at least among executives. Depending on the culture, this can cause interesting effects through internal competition. As an initiative matures and activities become more distributed, the SSG uses the centralized reporting system to keep track of all of the moving parts.

[SM3.2: 7] Run an external marketing program.

The SSG helps the firm market the software security initiative outside to build external support. Software security grows beyond being a risk reduction exercise and becomes a competitive advantage or market differentiator. The SSG might write papers or books about its SSDL. It might have a public blog. It might participate in external conferences or trade shows. In some cases, a complete SSDL methodology can be published and promoted externally. Sharing details externally and inviting critique can bring new perspectives into the firm.

Sharing details externally and inviting critique can bring new perspectives into the firm.



GOVERNANCE: Compliance & Policy (CP)

The Compliance & Policy practice is focused on identifying controls for compliance regimens such as PCI DSS and HIPAA, developing contractual controls such as service level agreements to help control COTS software risk, setting organizational software security policy, and auditing against that policy.

.....

CP LEVEL 1

[CP1.1: 45] Unify regulatory pressures.

If the business or its customers are subject to regulatory or compliance drivers such as FFIEC, GLBA, OCC, PCI

.....

DSS, SOX, HIPAA, or others, the SSG acts as a focal point for understanding the constraints such drivers impose on software. In some cases, the SSG creates a unified approach that removes redundancy from overlapping compliance requirements. A formal approach will map applicable portions of regulations to control statements explaining how the organization complies. As an alternative, existing business processes run by legal or other risk and compliance groups outside the SSG could also serve as the regulatory focal point. The goal of this activity is to create one set of software security guidance so that compliance work is completed as efficiently as possible (mostly by removing duplicates). Some firms move on to guide exposure by becoming directly involved in standards groups in order to influence the regulatory environment.

[CP1.2: 61] Identify PII obligations.

The way software handles personally identifiable information (PII) could be explicitly regulated, but even if it isn't, privacy is a hot topic. The SSG plays a key role in identifying and describing PII obligations stemming from regulation and customer expectations. It uses this information to promote best practices related to privacy. For example, if the organization processes credit card transactions, the SSG will identify the constraints that the PCI DSS places on the handling of cardholder data and inform all stakeholders. Note that outsourcing to hosted environments (e.g., the cloud) does not relax a majority of PII obligations. Also note, firms that create software products that process PII (but who don't necessarily handle PII directly) can get credit by providing privacy controls and guidance for their customers.

[CP1.3: 41] Create policy.

The SSG guides the rest of the organization by creating or contributing to software security policy that satisfies regulatory and customer-driven security requirements. The policy provides a unified approach for satisfying the (potentially lengthy) list of security drivers at the governance level. As a result, project teams can avoid keeping up with the details involved in complying with all applicable regulations. Likewise, project teams don't need to re-learn customer security requirements on their own. The SSG policy documents are sometimes focused around major compliance topics such as the handling of PII or the use of cryptography. In some cases, policy documents relate directly to the SSDL and its use in the firm. Architecture standards and coding guidelines are not examples of software security policy. On the other hand, policy that prescribes and mandates the use of coding guidelines and architecture standards for certain categories of applications does count. Policy is what is permitted and denied at the initiative level. If it's not mandatory, it's not policy.

**If it's not mandatory,
it's not policy.**

.....

CP LEVEL 2

[CP2.1: 19] Identify PII data inventory.

The organization identifies the kinds of PII stored by each of its systems and their data repositories. A PII inventory can be approached in two ways: starting with each individual application by noting its PII use or starting with particular types of PII and the applications that touch them. In either case, an inventory of data repositories is required. When combined with the organization's PII obligations, this inventory guides privacy planning. For example, the SSG can now create a list of databases that would require customer notification if breached.

.....

[CP2.2: 23] Require security sign-off for compliance-related risk.

The organization has a formal compliance risk acceptance and accountability process addressing all software development projects. The SSG might act as an advisor when the risk acceptor signs off on the state of the software prior to release. For example, the sign-off policy might require the head of the business unit to sign off on compliance issues that have not been mitigated or SSDL steps related to compliance that have been skipped. Sign-off should be explicit and captured for future reference. Any exceptions should be tracked.

[CP2.3: 25] Implement and track controls for compliance.

The organization can demonstrate compliance with applicable regulations because its SSDL is aligned with the control statements developed by the SSG (see [CP1.1 Unify regulatory pressures]). The SSG tracks the controls, shepherds problem areas, and makes sure auditors and regulators are satisfied. If the organization's SDLC is predictable and reliable, the SSG might be able to largely sit back and keep score. If the SDLC is uneven or less reliable, the SSG could be forced to take a more active role as referee. A firm doing this properly can explicitly associate satisfying its compliance concerns to following its SSDL.

[CP2.4: 29] Paper all vendor contracts with software security SLAs.

Vendor contracts include a service-level agreement (SLA) ensuring that the vendor will not jeopardize the organization's compliance story and software security initiative. Each new or renewed contract contains a set of provisions requiring the vendor to address software security and deliver a product or service compatible with the organization's security policy (see [SR2.5 Create SLA boilerplate]). In some cases, open source licensing concerns initiate the vendor control process. That can open the door for further software security language in the SLA. Traditional IT security requirements and a simple agreement to allow penetration testing are not sufficient.

[CP2.5: 33] Ensure executive awareness of compliance and privacy obligations.

The SSG gains executive buy-in around compliance and privacy activities. Executives are provided plain-language explanations of the organization's compliance and privacy obligations and the potential consequences for failing to meet those obligations. For some organizations, explaining the direct cost and likely fallout from a data breach could be an effective way to broach the subject. For other organizations, having an outside expert address the Board works because some executives value outside perspective more than internal perspective. One sure sign of proper executive awareness is adequate allocation of resources to get the job done. Be aware that the light and heat typically following a breach will not last.

.....

CP LEVEL 3

[CP3.1: 18] Create regulator eye candy.

The SSG has the information regulators want. A combination of written policy, controls documentation, and artifacts gathered through the SSDL gives the SSG the ability to demonstrate the organization's compliance story without a fire drill for every audit. In some cases, regulators, auditors, and senior management are satisfied with the same kinds of reports, which may be generated directly from various tools.

[CP3.2: 11] Impose policy on vendors.

Vendors are required to adhere to the same policies used internally. Vendors must submit evidence that their software security practices pass muster. Evidence could include results

**The SSG has
the information
regulators want.**

.....

from code reviews or penetration tests. Vendors may also attest to the fact that they are carrying out certain SSDL processes. In some cases, a BSIMM score or a vBSIMM score has been used to help ensure that vendors are complying with the firm's policies.

[CP3.3: 6] Drive feedback from SSDL data back to policy.

Information from the SSDL is routinely fed back into the policy creation process. Policies are improved to find defects earlier or prevent them from occurring in the first place. Blind spots are eliminated based on trends in SSDL failures. For example, inadequate architecture analysis, recurring vulnerabilities, ignored security gates, or choosing the wrong firm to carry out a penetration test may expose policy weakness. Over time, policies should become more practical and easier to carry out (see [SM1.1 *Publish process (roles, responsibilities, plan), evolve as necessary*]). Ultimately, policies align themselves with the SSDL data and enhance and improve a firm's effectiveness.



GOVERNANCE: Training (T)

Training has always played a critical role in software security because software developers and architects often start with very little security knowledge.

.....

T LEVEL 1

[T1.1: 59] Provide awareness training.

The SSG provides awareness training in order to promote a culture of software security throughout the organization. Training might be delivered by SSG members, by an outside firm, by the internal training organization, or through eLearning. Course content isn't necessarily tailored for a specific audience. For example, all programmers, quality assurance engineers, and project managers could attend the same 'Introduction to Software Security' course. This common activity can be enhanced with a tailored approach to an introductory course that addresses a firm's culture explicitly. Generic introductory courses covering basic IT security and high-level software security concepts do not generate satisfactory results. Likewise, providing awareness training only to developers and not to other roles is also insufficient.

[T1.5: 26] Deliver role-specific advanced curriculum (tools, technology stacks, and bug parade).

Software security training goes beyond building awareness and enables trainees to incorporate security practices into their work. The training is tailored to the role of trainees; trainees get information about the tools, technology stacks, or kinds of bugs that are most relevant to them. An organization might offer four tracks for engineers: one for architects, one for Java developers, one for Ruby developers, and a fourth for testers. Tool-specific training is also commonly observed in a curriculum. Don't forget that training will be useful for many different roles in an organization, including QA, product management, executives, and others.

[T1.6: 17] Create and use material specific to company history.

In order to make a strong and lasting change in behavior, training includes material specific to the company's history. When participants can see themselves in the problem, they are more likely to understand how the material is relevant to their work and to know when and how to apply what they have learned. One way to do this is to use noteworthy attacks on the company as examples in the training curriculum. Be wary of training that covers platforms not used by developers (Windows developers don't care about old Unix problems) or examples of problems only relevant to languages no longer in common use (Java developers don't need to understand buffer overflows in C). Stories from company history can help steer training in the right direction only if the stories are still relevant and not overly censored.

.....

[T1.7: 36] Deliver on-demand individual training.

The organization lowers the burden on trainees and reduces the cost of delivering training by offering on-demand training for individuals across roles. eLearning is the most obvious choice and can be kept up-to-date through a subscription model. Online courses must be engaging and relevant to achieve their intended purpose. For developers, it is also possible to provide training directly through IDEs right at the time it's needed. Remember that in some cases, building a new skill (such as code review) could be better suited for instructor-led training. Of course, training that sits around on the shelf does nobody any good.

.....

T LEVEL 2

[T2.5: 10] Enhance satellite through training and events.

The SSG strengthens the social network by holding special events for the satellite. The satellite learns about advanced topics or hears from guest speakers. Offering pizza and beer doesn't hurt. A standing conference call with voluntary attendance does not address this activity, which is as much about building camaraderie as it is about sharing knowledge or organizational efficiency.

There's no substitute for face-to-face meetings, even if they happen only once or twice a year.

**There's no substitute for
face-to-face meetings**

[T2.6: 15] Include security resources in onboarding.

The process for bringing new hires into the engineering organization requires a module about software security. The generic new hire process covers things like picking a good password and making sure people don't tail you into the building, but this can be enhanced to cover topics such as secure coding, the SSDL, and internal security resources. The objective is to ensure that new hires enhance the security culture. Turnover in engineering organizations is generally high. Though a generic onboarding module is useful, it does not take the place of a timely and more complete introductory software security course.

[T2.7: 6] Identify satellite through training.

The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill. Identifying this group is a step towards creating a social network that speeds the adoption of security into software development. One way to begin is to track the people who stand out during training courses (see [SM2.3 *Create or grow a satellite*]). In general, a volunteer army may be easier to lead than one that is drafted.

.....

T LEVEL 3

[T3.1: 3] Reward progression through curriculum (certification or HR).

Knowledge is its own reward, but progression through the security curriculum brings other benefits too. Developers, testers, and others see a career advantage in learning about security. The reward system can be formal and lead to a certification or official mark in the HR system, or it can be less formal and use motivators such as praise letters for the satellite written at annual review time. Involving a corporate training department and/or HR can make security's impact on career progression more obvious, but the SSG should continue to monitor security knowledge in the firm and not cede complete control or oversight.

.....

[T3.2: 3] Provide training for vendors or outsourced workers.

Spending time and effort helping suppliers get security right at the outset is easier than trying to figure out what they screwed up later on. In the best case, outsourced workers receive the same training given to employees. Training individual contractors is much more natural than training entire outsource firms and is a reasonable way to start. Of course, it's important to train everyone who works on your software regardless of their employment status.

[T3.3: 3] Host external software security events.

The organization highlights its security culture as a differentiator by hosting security events featuring external speakers and content. Good examples of this are Microsoft's BlueHat and Intel's Security Conference. Employees benefit from hearing outside perspectives. The organization as a whole benefits from putting its security cred on display (see [SM3.2 Run an external marketing program]). Events open to just certain small groups will not result in the desired change.

[T3.4: 8] Require an annual refresher.

Everyone involved in the SSDL is required to take an annual software security refresher course. The refresher keeps the staff up-to-date on security and ensures the organization doesn't lose focus due to turnover. The SSG might use half a day to give an update on the security landscape and explain changes to policies and standards. A refresher can be rolled out as part of a firm-wide security day or in concert with an internal security conference.

[T3.5: 4] Establish SSG office hours.

The SSG offers help to any and all comers during an advertised lab period or regularly scheduled office hours. By acting as an informal resource for people who want to solve security problems, the SSG leverages teachable moments and emphasizes the carrot over the stick. Office hours might be held one afternoon per week in the office of a senior SSG member. Mobile office hours are also a possibility, with visits to particular product or application groups slated by request.



INTELLIGENCE: Attack Models (AM)

Attack Models capture information used to think like an attacker: threat modeling, abuse case development and refinement, data classification, and technology-specific attack patterns.

.....

AM LEVEL 1

[AM1.1: 17] Build and maintain a top N possible attacks list.

The SSG helps the organization understand attack basics by maintaining a living list of attacks most important to the firm and using it to drive change. This list combines input from multiple sources: observed attacks, hacker forums, industry trends, etc. The list does not need to be updated with great frequency and the attacks can be sorted in a coarse fashion. For example, the SSG might brainstorm twice per year to create lists of attacks the organization should be prepared to counter "now," "soon," and "someday." In some cases, attack model information is used in a list-based approach to architecture analysis, helping to focus the analysis as in the case of [STRIDE](#).

[AM1.2: 51] Create a data classification scheme and inventory.

The organization agrees upon a data classification scheme and uses the scheme to inventory its software according to the kinds of data the software handles. This allows applications to be prioritized by their data classification. Many classification schemes are possible—one approach is to focus on PII. Depending upon the scheme and the software involved, it could be easiest to first classify data repositories, then derive classifications for applications according

.....

to the repositories they use. Other approaches to the problem are possible. For example, data could be classified according to protection of intellectual property, impact of disclosure, exposure to attack, relevance to SOX, or geographic boundaries.

[AM1.3: 31] Identify potential attackers.

The SSG identifies potential attackers in order to understand their motivations and capabilities. The outcome of this exercise could be a set of attacker profiles including generic sketches for categories of attackers and more detailed descriptions for noteworthy individuals. In some cases, a third-party vendor might be contracted to provide this information. Specific and contextual attacker information is almost always more useful than generic information copied from someone else's list.

[AM1.4: 8] Collect and publish attack stories.

To maximize the benefit from lessons that don't always come cheap, the SSG collects and publishes stories about attacks against the organization. Over time, this collection helps the organization understand its history. Both successful and unsuccessful attacks can be noteworthy. Discussing historical information about software attacks has the effect of grounding software security in the reality of a firm. This is particularly useful in training classes to counter a generic approach over-focused on top 10 lists or irrelevant and outdated platform attacks. Hiding information about attacks from people building new systems does nothing to garner positive benefit from a negative happenstance.

[AM1.5: 46] Gather and use attack intelligence.

The SSG stays ahead of the curve by learning about new types of attacks and vulnerabilities. The information comes from attending conferences and workshops, monitoring attacker forums, and reading relevant publications, mailing lists, and blogs. Make Sun Tzu proud by knowing your enemy; engage with the security researchers who are likely to cause you trouble. In many cases, a subscription to a commercial service provides a reasonable way of gathering basic attack intelligence. Regardless of its origin, attack information must be made actionable and useful for software builders and testers.

**Make Sun Tzu proud
by knowing your
enemy.**

[AM1.6: 11] Build an internal forum to discuss attacks.

The organization has an internal forum where the SSG, the satellite, and others discuss attacks. The forum serves to communicate the attacker perspective. The SSG could maintain an internal mailing list where subscribers share the latest information on publicly known incidents. Dissection of attacks and exploits that are relevant to a firm are particularly helpful when they spur discussion of development mitigations. Simply republishing items from public mailing lists doesn't achieve the same benefits as active discussion. Vigilance means never getting too comfortable (see [SR1.2 Create a security portal]).

.....

AM LEVEL 2

[AM2.1: 6] Build attack patterns and abuse cases tied to potential attackers.

The SSG prepares for security testing and architecture analysis by building attack patterns and abuse cases tied to potential attackers. These resources don't have to be built from scratch for every application in order to be useful. Instead, there could be standard sets for applications with similar profiles. The SSG will add to the pile based on attack stories. For example, a story about an attack against poorly managed entitlements could lead to an

.....

entitlements attack pattern that drives a new type of testing. If a firm tracks fraud and monetary costs associated with particular attacks, this information can be used to guide the process of building attack patterns and abuse cases.

[AM2.2: 8] Create technology-specific attack patterns.

The SSG creates technology-specific attack patterns to capture knowledge about attacks that target particular technologies. For example, if the organization's web software relies on cutting-edge browser capabilities, the SSG could catalogue the quirks of all the popular browsers and how they might be exploited. Attack patterns directly related to the security frontier (e.g., mobile security and wearable computing) can be useful. Simply republishing general guidelines (e.g., "Ensure data are protected in transit") and adding "for mobile applications" on the end does not constitute technology-specific attack patterns.

.....

AM LEVEL 3

[AM3.1: 4] Have a science team that develops new attack methods.

The SSG has a science team that works to identify and defang new classes of attacks before real attackers even know they exist. This isn't a penetration testing team finding new instances of known types of weaknesses—it's a research group innovating new types of attacks. A science team may include well-known security researchers who publish their findings at conferences like [Def Con](#).

[AM3.2: 2] Create and use automation to do what attackers will do.

The SSG arms testers and auditors with automation to do what attackers are going to do. For example, a new attack method identified by the science team could require a new tool. The SSG packages the new tool and distributes it to testers. The idea here is to push attack capability past what typical commercial tools and offerings encompass and then package that information for others to use. Tailoring these new tools to a firm's particular technology stacks and potential attackers is a really good idea.

INTELLIGENCE: Security Features & Design (SFD)



The Security Features & Design practice is charged with creating usable security patterns for major security controls (meeting the standards defined in the Standards & Requirements practice), building middleware frameworks for those controls, and creating and publishing other proactive security guidance.

SFD LEVEL 1

[SFD1.1: 61] Build and publish security features.

Some problems are best solved only once. Rather than have each project team implement all of their own security features (e.g., authentication, role management, key management, audit/log, cryptography, protocols), the SSG provides proactive guidance by building and publishing security features for other groups to use. Project teams benefit from implementations that come pre-approved by the SSG and the SSG benefits by not having to repeatedly track down the kinds of subtle errors that often creep into security features. The SSG can identify an implementation they like and promote it as the accepted solution.

Tailoring tools to a firm's particular technology stacks and potential attackers is a really good idea.

.....

[SFD1.2: 59] Engage SSG with architecture.

Security is a regular part of the organization's software architecture discussion. The architecture group takes responsibility for security the same way they take responsibility for performance, availability or scalability. One way to keep security from falling out of the discussion is to have an SSG member attend regular architecture meetings. In

other cases, enterprise architecture can help the SSG create secure designs that integrate properly into corporate design standards. Proactive engagement by the SSG is key to success.

Some problems are best solved only once.

SFD LEVEL 2

[SFD2.1: 24] Build secure-by-design middleware frameworks and common libraries.

The SSG takes a proactive role in software design by building or providing pointers to secure-by-design middleware frameworks or common libraries. In addition to teaching by example, this middleware aids architecture analysis and code review because the building blocks make it easier to spot errors. For example, the SSG could modify a popular web framework, such as Spring, to make it easy to meet input validation requirements. Eventually the SSG can tailor code review rules specifically for the components it offers (see [CR3.1 Use automated tools with tailored rules]). When adopting a middleware framework (or any other widely used software), careful vetting for security before publication is important. Encouraging adoption and use of insecure middleware does not help the software security situation. Generic open source software security architectures, including OWASP ESAPI, should not be considered secure by design. Bolting security on at the end by calling a library is not the way to approach secure design.

Bolting security on at the end...is not the way to approach secure design.

[SFD 2.2: 39] Create SSG capability to solve difficult design problems.

When the SSG is involved early in the new project process, it contributes to new architecture and solves difficult design problems. The negative impact security has on other constraints (time to market, price, etc.) is minimized. If a skilled security architect from the SSG is involved in the design of a new protocol, he or she could analyze the security implications of existing protocols and identify elements that should be duplicated or avoided.

Designing for security up front is more efficient than analyzing an existing design for security and then refactoring when flaws are uncovered. Some design problems will require specific expertise outside of the SSG.

SFD LEVEL 3

[SFD3.1: 8] Form a review board or central committee to approve and maintain secure design patterns.

A review board or central committee formalizes the process for reaching consensus on design needs and security tradeoffs. Unlike the architecture committee, this group is specifically focused on providing security guidance. The group also periodically reviews already-published design standards (especially around cryptography) to ensure that design decisions do not become stale or out of date.

[SFD3.2: 11] Require use of approved security features and frameworks.

Implementers must take their security features and frameworks from an approved list. There are two benefits: developers do not spend time re-inventing existing capabilities and review teams do not have to contend with finding the same old defects in brand new projects. In particular, the more a project uses proven components, the easier architecture analysis and code review become (see [AA1.1 Perform security feature review]). Re-use is a major advantage of consistent software architecture.

[SFD3.3: 2] Find and publish mature design patterns from the organization.

The SSG fosters centralized design reuse by collecting design patterns from across the organization and publishing them for everyone to use. A section of the SSG website could promote positive elements identified during architecture analysis so that good ideas are spread. This process should be formalized. An ad hoc, accidental noticing is not sufficient. In some cases, a central architecture or technology team facilitates and enhances this activity.



INTELLIGENCE: Standards & Requirements (SR)

The Standards & Requirements practice involves eliciting explicit security requirements from the organization, determining which COTS to recommend, building standards for major security controls (such as authentication, input validation and so on), creating security standards for technologies in use, and creating a standards review board.

SR LEVEL 1

[SR1.1: 57] Create security standards.

Software security requires much more than security features, but security features are part of the job as well. The SSG meets the organization's demand for security guidance by creating standards that explain the accepted way to adhere to policy and carry out specific security-centric operations. A standard might describe how to perform authentication using J2EE or how to determine the authenticity of a software update (see [SFD1.1 Build and publish security features] for one case where the SSG provides a reference implementation of a security standard). Standards can be deployed in a variety of ways. In some cases, standards and guidelines can be automated in development environments (e.g., worked into an IDE). In other cases, guidelines can be explicitly linked to code examples to make them more actionable and relevant. Standards that are not widely adopted and enforced are not really standards.

[SR1.2: 50] Create a security portal.

The organization has a well-known central location for information about software security. Typically, this is an internal website maintained by the SSG. People refer to the site for the latest and greatest on security standards and requirements as well as other resources provided by the SSG. An interactive wiki is better than a static portal with guideline documents that rarely change. Organizations can supplement these materials with mailing lists and face-to-face meetings.

Standards that are not widely adopted and enforced are not really standards.

[SR1.3: 52] Translate compliance constraints to requirements.

Compliance constraints are translated into software requirements for individual projects. This is a linchpin in the organization's compliance strategy—by representing compliance constraints explicitly with requirements, demonstrating compliance becomes a manageable task. For example, if the organization routinely builds software that processes credit card transactions, PCI DSS compliance could play a role in the SSDL during the requirements phase. In other cases, technology standards built for international interoperability reasons can include security guidance. Representing these standards as requirements helps with traceability and visibility in the case of audit.

.....

SR LEVEL 2

[SR2.2: 27] Create a standards review board.

The organization creates a standards review board to formalize the process used to develop standards and ensure that all stakeholders have a chance to weigh in. The review board could operate by appointing a champion for any proposed standard. The onus is on the champion to demonstrate that the standard meets its goals and to get approval and buy-in from the review board. Enterprise architecture or enterprise risk groups sometimes take on the responsibility of creating and managing standards review boards.

[SR2.3: 21] Create standards for technology stacks.

The organization standardizes on specific technology stacks. For the SSG, this means a reduced workload because the group does not have to explore new technology risks for every new project. Ideally, the organization will create a secure base configuration for each technology stack, further reducing the amount of work required to use the stack safely. A stack might include an operating system, a database, an application server, and a runtime environment for a managed language. The security frontier is a good place to find traction. Currently, mobile technology stacks and platforms as well as cloud-based technology stacks are two areas where specific attention to security pays off.

[SR2.4: 19] Identify open source.

The first step toward managing risk introduced by open source is to identify the open source components in use across the portfolio. It's not uncommon to discover old versions of components with known vulnerabilities or multiple versions of the same component. Automated tools for finding open source, whether whole components or large chunks of borrowed code, are one way to approach this activity. A process that relies solely on developers asking for permission does not generate satisfactory results. At the next level of maturity, this activity is subsumed by a policy constraining the use of open source.

[SR2.5: 20] Create SLA boilerplate.

The SSG works with the legal department to create a standard SLA boilerplate that is used in contracts with vendors and outsource providers to require software security efforts. The legal department understands that the boilerplate also helps prevent compliance and privacy problems. Under the agreement, vendors and outsource providers must meet company software security standards (see [CP2.4 *Paper all vendor contracts with software security SLAs*]). Boilerplate language may call out software security vendor control solutions such as vBSIMM measurements or BSIMM scores.

[SR2.6: 23] Use secure coding standards.

Secure coding standards help developers avoid the most obvious bugs and provide ground rules for code review. Secure coding standards are necessarily specific to a programming language and can address the use of popular frameworks and libraries. If the organization already has coding standards for other purposes, the secure coding standards should build upon them. A clear set of secure coding standards is a good way to guide both manual and automated code review, as well as beefing up security training with relevant examples. Remember, guidance does not a standard make.

.....

SR LEVEL 3

[SR3.1: 6] Control open source risk.

The organization has control over its exposure to the vulnerabilities that come along with using open source components. Use of open source could be restricted to pre-defined projects. It could also be restricted to open source versions that have been through an SSG security screening process, had unacceptable vulnerabilities remediated, and made available only through internal repositories. Legal often spearheads additional open source controls due to the “viral” license problem associated with GPL code. Getting legal to understand security risks can help move an organization to practice decent open source hygiene. Of course, this control must be applied across the software portfolio.

[SR3.2: 11] Communicate standards to vendors.

The SSG works with vendors to educate them and promote the organization’s security standards. A healthy relationship with a vendor cannot be guaranteed through contract language alone. The SSG engages with vendors, discusses the vendor’s security practices and explains in concrete terms (rather than legalese) what the organization expects of the vendor. Any time a vendor adopts the organization’s security standards, it’s a clear win. When a firm’s SSDL is available publically, communication regarding software security expectations is easier. Likewise, sharing internal practices and measures can make expectations very clear.



SSDL TOUCHPOINTS: Architecture Analysis (AA)

Architecture Analysis encompasses capturing software architecture in concise diagrams, applying lists of risks and threats, adopting a process for review (such as STRIDE or Architecture Risk Analysis), and building an assessment and remediation plan for the organization.

AA LEVEL 1

[AA1.1: 67] Perform security feature review.

To get started in architecture analysis, center the process on a review of security features.

Security-aware reviewers first identify the security features in an application (authentication, access control, use of cryptography, etc.) then study the design looking for problems that would cause these features to fail at their purpose or otherwise prove insufficient. For example, a system that was subject to escalation of privilege attacks because of broken access control or a system that stored unsalted password hashes would both be identified in this kind of review. At higher levels of maturity, the activity of reviewing features is eclipsed by a more thorough approach to architecture analysis. In some cases, use of the firm’s secure-by-design components can streamline this process.

**Sharing internal practices
and measures can make
expectations very clear.**

[AA1.2: 29] Perform design review for high-risk applications.

The organization learns about the benefits of architecture analysis by seeing real results for a few high-risk, high-profile applications. The reviewers must have some experience performing detailed design review and breaking the architecture being considered. In all cases, design review produces a set of architecture flaws and a plan to mitigate

them. If the SSG is not yet equipped to perform an in-depth architecture analysis, it uses consultants to do this work. Ad hoc review paradigms that rely heavily on expertise can be used here, though in the long run they do not scale. A review focused only on whether a software project has performed the right process steps will not generate expected results.

[AA1.3: 22] Have SSG lead design review efforts.

The SSG takes a lead role in architecture analysis by performing design review to build the organization's ability to uncover design flaws. Breaking an architecture is enough of an art that the SSG must be proficient at it before they can turn the job over to the architects, and proficiency requires practice. The SSG cannot be successful on its own either—it's likely they'll need help from architects or implementers to understand the design. With a clear design in hand, the SSG might carry out the detailed review with a minimum of interaction with the project team. At higher levels of maturity, the responsibility for leading review efforts shifts towards software architects. Approaches to architecture analysis (and threat modeling) evolve over time. Do not expect to set a process and use it forever.

[AA1.4: 46] Use a risk questionnaire to rank applications.

To facilitate security feature and design review processes, the SSG uses a risk questionnaire to collect basic information about each application so that it can determine a risk classification and prioritization scheme. Questions might include, "Which programming languages is the application written in?" "Who uses the application?" and "Does the application handle PII?" A qualified member of the application team completes the questionnaire. The questionnaire is short enough to be completed in a matter of hours. The SSG might use the answers to bucket the application as high, medium, or low risk. Because a risk questionnaire can be easy to game, it's important to put some spot checking for validity and accuracy into place. An over-reliance on self-reporting or automation can render this activity impotent.

Do not expect to set a process and use it forever.

.....

AA LEVEL 2

[AA2.1: 12] Define and use AA process.

The SSG defines and documents a process for architecture analysis and applies it in the design reviews it conducts. The process includes a standardized approach for thinking about attacks and security properties, and the associated risk. The process is defined rigorously enough that people outside the SSG can be taught to carry it out. Particular attention should be paid to documentation of both the architecture under review and any security flaws uncovered. Tribal knowledge doesn't count as a defined process. Microsoft's STRIDE and [Digital's ARA](#) are examples of this process. Note that even these two methodologies for architecture analysis have evolved greatly over time. Make sure to access up-to-date sources for architecture analysis information because many early publications are outdated and no longer apply.

[AA2.2: 9] Standardize architectural descriptions (including data flow).

AA processes use an agreed-upon format for describing architecture, including a means for representing data flow. This format, combined with an architecture analysis process, makes architecture analysis tractable for people who are not security experts. A standard architecture description can be enhanced to provide an explicit picture of information assets that require protection. Standardized icons that are consistently used in UML diagrams, Visio templates, and whiteboard squiggles are especially useful.

.....

[AA2.3: 13] Make SSG available as AA resource or mentor.

To build an architecture analysis capability outside of the SSG, the SSG advertises itself as a resource or mentor for teams who ask for help using the AA process to conduct their own design review and proactively seek projects to get involved with. The SSG will answer architecture analysis questions during office hours, and in some cases, might assign someone to sit side-by-side with the architect for the duration of the analysis. In the case of high-risk applications or products, the SSG plays a more active mentorship role in applying the AA process.

AA LEVEL 3

[AA3.1: 6] Have software architects lead design review efforts.

Software architects throughout the organization lead the architecture analysis process most of the time. The SSG still might contribute to architecture analysis in an advisory capacity or under special circumstances. This activity requires a well-understood and well-documented architecture analysis process. Even in that case, consistency is very difficult to attain because breaking architectures requires so much experience.

[AA3.2: 1] Drive analysis results into standard architecture patterns.

Failures identified during architecture analysis are fed back to the security design committee so that similar mistakes can be prevented in the future through improved design patterns (see [SFD3.1 Form a review board or central committee to approve and maintain secure design patterns]). Security design patterns can interact in surprising ways that break security. The architecture analysis process should be applied even when vetted design patterns are in standard use.



SSDL TOUCHPOINTS: Code Review (CR)

The Code Review practice includes use of code review tools, development of tailored rules, customized profiles for tool use by different roles (for example, developers versus auditors), manual analysis, and tracking/measuring results.

CR LEVEL 1

[CR1.1: 18] Use a top N bugs list (real data preferred).

The SSG maintains a list of the most important kinds of bugs that must be eliminated from the organization's code and uses it to drive change. The list helps focus the organization's attention on the bugs that matter most. It's okay to start with a generic list pulled from public sources, but a list is much more valuable if it's specific to the organization and built from real data gathered from code review, testing, and actual incidents. The SSG can periodically update the list and publish a "most wanted" report. (For another way to use the list, see [T1.6 Create and use material specific to company history]). Some firms use multiple tools and real code base data to build Top N lists, not constraining themselves to a particular service or tool. One potential pitfall with a Top N list is the problem of "looking for your keys only

The OWASP Top 10 list rarely reflects an organization's bug priorities.

under the street light.” For example, the OWASP Top 10 list rarely reflects an organization’s bug priorities. Simply sorting the day’s bug data by number of occurrences doesn’t produce a satisfactory Top N list because these data change so often.

[CR1.2: 53] Have SSG perform ad hoc review.

The SSG performs an ad hoc code review for high-risk applications in an opportunistic fashion. For example, the SSG might follow up the design review for high-risk applications with a code review. At higher maturity levels, replace ad hoc targeting with a systematic approach. SSG review could involve the use of specific tools and services, or it might be manual.

[CR1.4: 55] Use automated tools along with manual review.

Incorporate static analysis into the code review process to make code review more efficient and more consistent. The automation doesn’t replace human judgment, but it does bring definition to the review process and security expertise to reviewers who are not security experts. A firm may use an external service vendor as part of a formal code review process for software security. This service should be explicitly connected to a larger SSDL applied during software development and not just “check the security box” on the path to deployment.

[CR1.5: 24] Make code review mandatory for all projects.

Code review is a mandatory release gate for all projects under the SSG’s purview. Lack of code review or unacceptable results will stop the release train. While all projects must undergo code review, the review process might be different for different kinds of projects. The review for low-risk projects might rely more heavily on automation and the review for high-risk projects might have no upper bound on the amount of time spent by reviewers. In most cases, a code review gate with a minimum acceptable standard forces projects that don’t pass to be fixed and re-evaluated before they ship.

[CR1.6: 27] Use centralized reporting to close the knowledge loop and drive training.

The bugs found during code review are tracked in a centralized repository. This repository makes it possible to do summary reporting and trend reporting for the organization. The SSG can use the reports to demonstrate progress and drive the training curriculum (see [SM2.5 *Identify metrics and use them drive budgets*]). Code review information can be incorporated into a CSO-level dashboard that includes feeds from other parts of the security organization. Likewise, code review information can be fed into a development-wide project tracking system that rolls up several diverse software security feeds (for example, penetration tests, security testing, black box testing, white box testing, etc.). Don’t forget that individual bugs make excellent training examples.

Individual bugs make excellent training examples.

CR LEVEL 2

[CR2.2: 7] Enforce coding standards.

A violation of the organization’s secure coding standards is sufficient grounds for rejecting a piece of code. Code review is objective—it shouldn’t devolve into a debate about whether or not bad code is exploitable. The enforced portion of the standard could start out being as simple as a list of banned functions. In some cases, coding standards for developers are published specific to technology stacks (for example, guidelines for C++ or Spring) and then enforced during the code review process or directly in the IDE. Standards can be positive (“do it this way”) as well as negative (“do not use this API”).

[CR2.5: 20] Assign tool mentors.

Mentors are available to show developers how to get the most out of code review tools. If the SSG is most skilled with the tools, it could use office hours to help developers establish the right configuration or get started interpreting results. Alternatively, someone from the SSG might work with a development team for the duration of the first review they perform. Centralized use of a tool can be distributed into the development organization over time through the use of tool mentors.

[CR2.6: 16] Use automated tools with tailored rules.

Customize static analysis to improve efficiency and reduce false positives. Use custom rules to find errors specific to the organization's coding standards or custom middleware. Turn off checks that aren't relevant. The same group that provides tool mentoring will likely spearhead the customization. Tailored rules can be explicitly tied to proper usage of technology stacks in a positive sense and avoidance of errors commonly encountered in a firm's code base in a negative sense.

.....

CR LEVEL 3

[CR3.2: 3] Build a factory.

Combine assessment results so that multiple analysis techniques feed into one reporting and remediation process. The SSG might write scripts to invoke multiple detection techniques automatically and combine the results into a format that can be consumed by a single downstream review and reporting solution. Analysis engines may combine static and dynamic analysis. The tricky part of this activity is normalizing vulnerability information from disparate sources that use conflicting terminology. In some cases, a CWE-like approach can help with nomenclature. Combining multiple sources helps drive better informed risk mitigation decisions.

[CR3.3: 5] Build a capability for eradicating specific bugs from the entire codebase.

When a new kind of bug is found, the SSG writes rules to find it and uses the rules to identify all occurrences of the new bug throughout the entire codebase. It's possible to eradicate the bug type entirely without waiting for every project to reach the code review portion of its lifecycle. A firm with only a handful of software applications will have an easier time with this activity than firms with a very large number of large apps.

[CR3.4: 3] Automate malicious code detection.

Automated code review is used to identify dangerous code written by malicious in-house developers or outsource providers. Examples of malicious code that could be targeted include backdoors, logic bombs, time bombs, nefarious communication channels, obfuscated program logic, and dynamic code injection. Although out-of-the-box automation might identify some generic malicious-looking constructs, custom rules for static analysis tools used to codify acceptable, and unacceptable code patterns in the organization's codebase will quickly become a necessity. Manual code review for malicious code is a good start, but is insufficient to complete this activity.

SSDL TOUCHPOINTS: Security Testing (ST)



The Security Testing practice is concerned with pre-release testing, including integrating security into standard quality assurance processes. The practice includes use of black box security tools (including fuzz testing) as a smoke test in QA, risk-driven white box testing, application of the attack model, and code coverage analysis. Security testing focuses on vulnerabilities in construction.

.....

.....

ST LEVEL 1

[ST1.1: 61] Ensure QA supports edge/boundary value condition testing.

The QA team goes beyond functional testing to perform basic adversarial tests. They probe simple edge cases and boundary conditions. No attacker skills required. When QA understands the value of pushing past standard functional testing using acceptable input, they begin to move slowly toward “thinking like a bad guy.” A discussion of boundary value testing leads naturally to the notion of an attacker probing the edges on purpose. What happens when you enter the wrong password over and over?

[ST1.3: 66] Drive tests with security requirements and security features.

Testers target declarative security mechanisms derived from requirements and security features. For example, a tester could try to access administrative functionality as an unprivileged user or verify that a user account becomes locked after some number of failed authentication attempts. For the most part, security features can be tested in a similar fashion to other software features. Security

mechanisms based on requirements such as account lockout, transaction limitations, entitlements, and so on are also tested. Of course, software security is not security software, but getting started with features is easy.

Security testing focuses on vulnerabilities in construction.

.....

ST LEVEL 2

[ST2.1: 24] Integrate black box security tools into the QA process.

The organization uses one or more black box security testing tools as part of the quality assurance process. The tools are valuable because they encapsulate an attacker’s perspective, albeit in a generic fashion. Tools such as IBM Security AppScan or HP WebInspect are relevant for web applications, and fuzzing frameworks such as Codenomicon are applicable for most network protocols. In some situations, other groups might collaborate with the SSG to apply the tools. For example, a testing team could run the tool, but come to the SSG for help interpreting the results. Regardless of who runs the black box tool, the testing should be properly integrated into the QA cycle of the SSDL.

[ST2.4: 8] Share security results with QA.

The SSG routinely shares results from security reviews with the QA department. Over time, QA engineers learn the security mindset. Using security results to inform and evolve particular testing patterns can be a powerful mechanism leading to better security testing. This activity benefits from an engineering-focused QA function that is highly technical.

[ST2.5: 10] Include security tests in QA automation.

Security tests run alongside functional tests as part of automated regression testing. The same automation framework houses both. Security testing is part of the routine. Security tests can be driven from abuse cases identified earlier in the lifecycle or tests derived from creative tweaks of functional tests.

.....

[ST2.6: 11] Perform fuzz testing customized to application APIs.

Test automation engineers customize a fuzzing framework to the organization's APIs. They could begin from scratch or use an existing fuzzing toolkit, but customization goes beyond creating custom protocol descriptions or file format templates. The fuzzing framework has a built-in understanding of the application interfaces it calls into. Test harnesses developed explicitly for particular applications can make good places to integrate fuzz testing.

.....

ST LEVEL 3

[ST3.3: 4] Drive tests with risk analysis results.

Testers use architecture analysis results to direct their work. For example, if architecture analysis concludes, "the security of the system hinges on the transactions being atomic and not being interrupted partway through," then torn transactions will become a primary target in adversarial testing. Adversarial tests like these can be developed according to risk profile—high-risk flaws first.

[ST3.4: 4] Leverage coverage analysis.

Testers measure the code coverage of their security tests to identify code that isn't being exercised. Code coverage drives increased security testing depth. Standard-issue black box testing tools achieve exceptionally low coverage, leaving a majority of the software under test unexplored. Don't let this happen to your tests. Using standard measurements for coverage such as function coverage, line coverage, or multiple condition coverage is fine.

[ST3.5: 5] Begin to build and apply adversarial security tests (abuse cases).

Testing begins to incorporate test cases based on abuse cases. Testers move beyond verifying functionality and take on the attacker's perspective. For example, testers might systematically attempt to replicate incidents from the organization's history. Abuse and misuse cases based on the attacker's perspective can also be driven from security policies, attack intelligence and guidelines. This turns the corner from testing features to attempting to break the software under test.



DEPLOYMENT: Penetration Testing (PT)

The Penetration Testing practice involves standard outside—>in testing of the sort carried out by security specialists. Penetration testing focuses on vulnerabilities in the final configuration and provides direct feeds to defect management and mitigation.

.....

PT LEVEL 1

[PT1.1: 69] Use external penetration testers to find problems.

Many organizations aren't willing to address software security until there's unmistakable evidence that the organization isn't somehow magically immune to the problem. If security has not been a priority, external penetration testers can demonstrate that the organization's code needs help. Penetration testers could be brought in to break a high-profile application to make the point. Over time, the focus of penetration testing moves from "I told you our stuff was broken" to a smoke test and sanity check done before shipping. External penetration testers bring a new set of eyes to the problem.

[PT1.2: 47] Feed results to the defect management and mitigation system.

Penetration testing results are fed back to development through established defect management or mitigation channels, and development responds using their defect management and release process. The exercise

.....

demonstrates the organization's ability to improve the state of security. Many firms are beginning to emphasize the critical importance of not just identifying but, more importantly, fixing security problems. One way to ensure attention is to add a security flag to the bug tracking and defect management system. Evolving DevOps and integrated team structures do not eliminate the need for formalized defect management systems.

[PT 1.3: 47] Use penetration testing tools internally.

The organization creates an internal penetration testing capability that uses tools. This capability can be part of the SSG or part of a specialized and trained team elsewhere in the organization. The tools improve efficiency and repeatability of the testing process. Tools can include off-the-shelf products, standard issue network penetration tools that understand the application layer, and hand-written scripts.

PT LEVEL 2

[PT2.2: 20] Provide penetration testers with all available information.

Penetration testers, whether internal or external, use all available information about their target. Penetration testers can do deeper analysis and find more interesting problems when they have source code, design documents, architecture analysis results, and code review results. Give

penetration testers everything you have created throughout the SSDL and ensure they use it. If your penetration tester doesn't ask for the code, you need a new penetration tester.

If your penetration tester doesn't ask for the code, you need a new penetration tester.

[PT2.3: 17] Schedule periodic penetration tests for application coverage.

Periodically test all applications in the SSG's purview according to an established schedule, which could be tied to the calendar or to the release cycle. The testing serves as a sanity check and helps ensure yesterday's software isn't vulnerable to today's attacks. High-profile applications might get a penetration test at least once a year. One important aspect of periodic testing is to make sure that the problems identified in a penetration test are actually fixed and they don't creep back into the build.

PT LEVEL 3

[PT3.1: 10] Use external penetration testers to perform deep-dive analysis.

The organization uses external penetration testers to do deep-dive analysis for critical projects and to introduce fresh thinking into the SSG. These testers are experts and specialists; they keep the organization up to speed with the latest version of the attacker's perspective and have a track record for breaking the type of software being tested. Skilled penetration testers will always break a system. The question is whether they demonstrate new kinds of thinking about attacks that can be useful when designing, implementing, and hardening new systems. Creating new types of attacks from threat intelligence and abuse cases prevents checklist-driven approaches that only look for known types of problems.

[PT3.2: 8] Have the SSG customize penetration testing tools and scripts.

The SSG either creates penetration testing tools or adapts publicly-available tools so they can more efficiently and comprehensively attack the organization's systems. Tools improve the efficiency of the penetration testing process without sacrificing the depth of problems the SSG can identify. Tools that can be tailored are always preferable to

generic tools. This activity considers both the depth of tests and their scope.



DEPLOYMENT: Software Environment (SE)

The Software Environment practice concerns itself with OS and platform patching, web application firewalls, installation and configuration documentation, application monitoring, change management and, ultimately, code signing.

.....

SE LEVEL 1

[SE1.1: 37] Use application input monitoring.

The organization monitors the input to software it runs in order to spot attacks. For web code, a web application firewall (WAF) can do the job. The SSG could be responsible for the care and feeding of the system. Incident response is not part of this activity. Defanged WAFs that write log files can be useful if somebody reviews the logs periodically. On the other hand, a WAF that's unmonitored makes no noise when an application falls in the woods.

[SE1.2: 69] Ensure host and network security basics are in place.

The organization provides a solid foundation for software by ensuring that host and network security basics are in place. It is common for operations security teams to be responsible for duties such as patching operating systems and maintaining firewalls. Doing software security before network security is like putting on your pants before putting on your underwear.

Doing software security before network security is like putting on your pants before putting on your underwear.

.....

SE LEVEL 2

[SE2.2: 31] Publish installation guides.

The SSDL requires the creation of an installation guide to help deployment teams and operators install and configure the software securely. If special steps are required to ensure a deployment is secure, the steps are outlined in the installation guide. The guide should include discussion of COTS components. In some cases, installation guides are distributed to customers who buy the software. Evolving DevOps and integrated team structures do not eliminate the need for written guides. Of course, secure by default is always the best way to go.

[SE2.4: 25] Use code signing.

The organization uses code signing for software published across trust boundaries. Code signing is particularly useful for protecting the integrity of software that leaves the organization's control, such as shrink-wrapped applications or thick clients. The fact that some mobile platforms require application code to be signed does not indicate institutional use of code signing.

.....

SE LEVEL 3

[SE3.2: 10] Use code protection.

To protect intellectual property and make exploit development harder, the organization erects barriers to reverse engineering. Obfuscation techniques could be applied as part of the production build and release process.

.....

Employing platform-specific controls such as Data Execution Prevention (DEP), Safe Structured Error Handling (SafeSEH), and Address Space Layout Randomization (ASLR) can make exploit development more difficult.

[SE3.3: 5] Use application behavior monitoring and diagnostics.

The organization monitors the behavior of production software looking for misbehavior and signs of attack. This activity goes beyond host and network monitoring to look for problems that are specific to the software, such as indications of fraud. Intrusion detection and anomaly detection systems at the application level may focus on an application's interaction with the operating system (through system calls) or with the kinds of data that an application consumes, originates, and manipulates.



DEPLOYMENT: Configuration Management & Vulnerability Management (CMVM)

The Configuration Management & Vulnerability Management practice concerns itself with patching and updating applications, version control, defect tracking and remediation, and incident handling.

CMVM LEVEL 1

[CMVM1.1: 71] Create or interface with incident response.

The SSG is prepared to respond to an incident and is regularly included in the incident response process. The group either creates its own incident response capability or regularly interfaces with the organization's existing incident response team. A regular meeting between the SSG and the incident response team can keep information flowing in both directions. In many cases, software security initiatives have evolved from incident response teams who began to realize that software vulnerabilities were the bane of their existence.

[CMVM1.2: 73] Identify software defects found in operations monitoring and feed them back to development.

Defects identified through operations monitoring are fed back to development and used to change developer behavior. The contents of production logs can be revealing (or can reveal the need for improved logging). In some cases, providing a way to enter incident triage data into an existing bug tracking system (many times making use of a special security flag) seems to work. The idea is to close the information loop and make sure security problems get fixed. In the best of cases, processes in the SSDL can be improved based on operational data.

CMVM LEVEL 2

[CMVM2.1: 64] Have emergency codebase response.

The organization can make quick code changes when an application is under attack. A rapid-response team works in conjunction with the application owners and the SSG to study the code and the attack, find a resolution, and push a patch into production. Often, the emergency response team is the development team itself. Fire drills don't count; a well-defined process is required. A process that has never been used may not actually work.

[CMVM2.2: 61] Track software bugs found in operations through the fix process.

Defects found in operations are fed back to development, entered into established defect management systems and tracked through the fix process. This capability could come in the form of a two-way bridge between the bug finders and the bug fixers. Make sure the loop is closed completely. Setting a security flag in the bug-tracking system can help facilitate tracking.

[CMVM2.3: 31] Develop an operations inventory of applications.

The organization has a map of its software deployments. If a piece of code needs to be changed, operations can reliably identify all of the places where the change needs to be installed. Sometimes common components shared between multiple projects are noted so that when an error occurs in one application, other applications that share the same components can be fixed as well.

Setting a security flag in the bug-tracking system can help facilitate tracking.

CMVM LEVEL 3

[CMVM3.1: 4] Fix all occurrences of software bugs found in operations.

The organization fixes all instances of each software bug found during operations and not just the small number of instances that have triggered bug reports. This requires the ability to reexamine the entire codebase when new kinds of bugs come to light (see [CR3.3 *Build capability for eradicating specific bugs from entire codebase*]). One way to approach this is to create a rule set that generalizes a deployed bug into something that can be scanned for using automated code review.

[CMVM3.2: 6] Enhance the SSDL to prevent software bugs found in operations.

Experience from operations leads to changes in the SSDL. The SSDL is strengthened to prevent the reintroduction of bugs found during operations. To make this process systematic, the incident response post mortem could include a “feedback to SSDL” step. This works best when root cause analysis pinpoints where in the SDLC an error could have been introduced or slipped by uncaught. An ad hoc approach is not sufficient.

[CMVM3.3: 6] Simulate software crisis.

The SSG simulates high-impact software security crises to ensure software incident response capabilities minimize damage. Simulations could test for the ability to identify and mitigate specific threats or, in other cases, could begin with the assumption that a critical system or service is already compromised and evaluate the organization’s ability to respond. When simulations model successful attacks, an important question to consider is the time period required to clean up. Regardless, simulations must focus on security-relevant software failure and not on natural disasters or other types of emergency response drills. If the data center is burning to the ground, the SSG won’t be among the first responders.

[CMVM3.4: 3] Operate a bug bounty program.

The organization solicits vulnerability reports from external researchers and pays a bounty for each verified and accepted vulnerability received. Payouts typically follow a sliding scale linked to multiple factors, such as vulnerability type (e.g., remote code execution is worth \$10,000 versus CSRF is worth \$750), exploitability (demonstrable exploits command much higher payouts), or specific services and software versions (widely-deployed or critical services warrant higher payouts). Ad hoc or short-duration activities, such as capture-the-flag contests, do not count.

APPENDIX

Adjusting BSIMM-V for BSIMM6

Because the BSIMM is a data driven model, we have chosen to make adjustments to the model based on the data observed between BSIMM-V and BSIMM6.

We have added, deleted, and adjusted the level of various activities based on the data observed as the project continues. To preserve backwards compatibility, all changes are made by adding new activity labels to the model, even when an activity has simply changed levels. We make changes by considering outliers both in the model itself and in the levels we assigned to various activities in the 12 practices. We use the results of an intra-level standard deviation analysis to determine which “outlier” activities to move between levels. We focus on changes that minimize standard deviation in the average number of observed activities at each level.

Here are the four changes we made according to that paradigm:

1. [SM1.6] became [SM2.6]; [SM1.6] was removed (level promotion)
2. [SR1.4] became [SR2.6]; [SR1.4] was removed (level promotion)
3. [ST3.1] became [ST2.5]; [ST3.1] was removed (level demotion)
4. [ST3.2] became [ST2.6]; [ST3.2] was removed (level demotion)

We also carefully considered, but did not adjust, the following activities [SM2.1][AM1.4][CR 1.1][CR2.2][ST2.4]. We are keeping a close eye on the Attack Models and Code Review practices; both of which were impacted by culling aged data.

112 BSIMM Activities at a Glance

(Red indicates most observed BSIMM activity in that practice)

Level 1 Activities



Governance

Strategy & Metrics (SM)

- Publish process (roles, responsibilities, plan), evolve as necessary. [SM1.1]
- Create evangelism role and perform internal marketing. [SM1.2]
- Educate executives. [SM1.3]
- **Identify gate locations, gather necessary artifacts. [SM1.4]**

Compliance & Policy (CP)

- Unify regulatory pressures. [CP1.1]
- **Identify PII obligations. [CP1.2]**
- Create policy. [CP1.3]

Training (T)

- **Provide awareness training. [T1.1]**
- Deliver role-specific advanced curriculum (tools, technology stacks and bug parade). [T1.5]
- Create and use material specific to company history. [T1.6]
- Deliver on-demand individual training. [T1.7]



Intelligence

Attack Models (AM)

- Build and maintain a top N possible attacks list. [AM1.1: 17]
- **Create a data classification scheme and inventory. [AM1.2]**
- Identify potential attackers. [AM1.3]
- Collect and publish attack stories. [AM1.4]
- Gather and use attack intelligence. [AM1.5]
- Build an internal forum to discuss attacks. [AM1.6]

Security Features & Design (SFD)

- **Build and publish security features. [SFD1.1]**
- Engage SSG with architecture. [SFD1.2]

Standards & Requirements (SR)

- **Create security standards. [SR1.1]**
- Create a security portal. [SR1.2]
- Translate compliance constraints to requirements. [SR1.3]



SSDL Touchpoints

Architecture Analysis (AA)

- **Perform security feature review. [AA1.1]**
- Perform design review for high-risk applications. [AA1.2]
- Have SSG lead design review efforts. [AA1.3]
- Use a risk questionnaire to rank applications. [AA1.4]

Code Review (CR)

- Use a top N bugs list (real data preferred). [CR1.1]
- Have SSG perform ad hoc review. [CR1.2]
- **Use automated tools along with manual review. [CR1.4]**
- Make code review mandatory for all projects. [CR1.5]
- Use centralized reporting to close the knowledge loop and drive training. [CR1.6]

Security Testing (ST)

- Ensure QA supports edge/boundary value condition testing. [ST1.1]
- **Drive tests with security requirements and security features. [ST 1.3]**



Deployment

Penetration Testing (PT)

- **Use external penetration testers to find problems. [PT1.1]**
- Feed results to the defect management and mitigation system. [PT1.2]
- Use penetration testing tools internally. [PT1.3]

Software Environment (SE)

- Use application input monitoring. [SE1.1]
- **Ensure host and network security basics are in place. [SE1.2]**

Configuration Management & Vulnerability Management (CMVM)

- Create or interface with incident response. [CMVM1.1]
- **Identify software defects found in operations monitoring and feed them back to development. [CMVM 1.2]**

Level 2 Activities



Governance

Strategy & Metrics (SM)

- Publish data about software security internally. [SM2.1]
- Enforce gates with measurements and track exceptions. [SM2.2]
- Create or grow a satellite. [SM2.3]
- Identify metrics and use them to drive budgets. [SM2.5]
- Require security sign-off. [SM2.6]

Compliance & Policy (CP)

- Identify PII data inventory. [CP2.1]
- Require security sign-off for compliance-related risk. [CP2.2]
- Implement and track controls for compliance. [CP2.3]
- Paper all vendor contracts with software security SLAs. [CP2.4]
- Ensure executive awareness of compliance and privacy obligations. [CP2.5]

Training (T)

- Enhance satellite through training and events. [T2.5]
- Include security resources in onboarding. [T2.6]
- Identify satellite through training. [T2.7]



Intelligence

Attack Models (AM)

- Build attack patterns and abuse cases tied to potential attackers. [AM2.1]
- Create technology-specific attack patterns. [AM2.2]

Security Features & Design (SFD)

- Build secure-by-design middleware frameworks and common libraries. [SFD2.1]
- Create SSG capability to solve difficult design problems. [SFD2.2]

Standards & Requirements (SR)

- Create a standards review board. [SR2.2]
- Create standards for technology stacks. [SR2.3]
- Identify open source. [SR2.4]
- Create a SLA boilerplate. [SR2.5]
- Use secure coding standards. [SR2.6]



SSDL Touchpoints

Architecture Analysis (AA)

- Define and use AA process. [AA2.1]
- Standardize architectural descriptions (including data flow). [AA2.2]
- Make SSG available as AA resource or mentor. [AA2.3]

Code Review (CR)

- Enforce coding standards. [CR2.2]
- Assign tool mentors. [CR2.5]
- Use automated tools with tailored rules. [CR2.6]

Security Testing (ST)

- Integrate black box security tools into the QA process. [ST2.1]
- Share security results with QA. [ST2.4]
- Include security tests in QA automation. [ST2.5]
- Perform fuzz testing customized to application APIs. [ST2.6]



Deployment

Penetration Testing (PT)

- Provide penetration testers with all available information. [PT2.2]
- Schedule periodic penetration tests for application coverage. [PT2.3]

Software Environment (SE)

- Publish installation guides. [SE2.2]
- Use code signing. [SE2.4]

Configuration Management & Vulnerability Management (CMVM)

- Have emergency codebase response. [CMVM2.1]
- Track software bugs found in operations through the fix process. [CMVM2.2]
- Develop an operations inventory of applications. [CMVM2.3]

Level 3 Activities



Governance

Strategy & Metrics (SM)

- Use an internal tracking application with portfolio view. [SM3.1]
- Run an external marketing program. [SM3.2]

Compliance & Policy (CP)

- Create regulator eye candy. [CP3.1]
- Impose policy on vendors. [CP3.2]
- Drive feedback from SSDL data back to policy. [CP3.3]

Training (T)

- Reward progression through curriculum (certification or HR). [T3.1]
- Provide training for vendors or outsourced workers. [T3.2]
- Host external software security events. [T3.3]
- Require an annual refresher. [T3.4]
- Establish SSG office hours. [T3.5]



Intelligence

Attack Models (AM)

- Have a science team that develops new attack methods. [AM3.1]
- Create and use automation to do what attackers will do. [AM3.2]

Security Features & Design (SFD)

- Form a review board or central committee to approve and maintain secure design patterns. [SFD 3.1]
- Require use of approved security features and frameworks. [SFD3.2]
- Find and publish mature design patterns from the organization. [SFD3.3]

Standards & Requirements (SR)

- Control open source risk. [SR3.1]
- Communicate standards to vendors. [SR3.2]



SSDL Touchpoints

Architecture Analysis (AA)

- Have software architects lead design review efforts. [AA3.1]
- Drive analysis results into standard architecture patterns. [AA3.2]

Security Testing (ST)

- Drive tests with risk analysis results. [ST3.3]
- Leverage coverage analysis. [ST3.4]
- Begin to build and apply adversarial security tests (abuse cases). [ST3.5]

Code Review (CR)

- Build a factory. [CR3.2]
- Build a capability for eradicating specific bugs from the entire codebase. [CR3.3]
- Automate malicious code detection. [CR3.4]



Deployment

Penetration Testing (PT)

- Use external penetration testers to perform deep-dive analysis. [PT3.1]
- Have the SSG customize penetration testing tools and scripts. [PT3.2]

Software Environment (SE)

- Use code protection. [SE3.2]
- Use application behavior monitoring and diagnostics. [SE3.3]

Configuration Management & Vulnerability Management (CMVM)

- Fix all occurrences of software bugs found in operations. [CMVM3.1]
- Enhance the SSDL to prevent software bugs found in operations. [CMVM3.2]
- Simulate software crisis. [CMVM3.3]
- Operate a bug bounty program. [CMVM3.4]



Interested in joining the growing BSIMM Community?

Go to www.BSIMM.com